FILED
DEC 15 2006
OFFICE OF THE CLERK
SUPREME COURT, U.S.

(28)

No. 05-1056

IN THE

# Supreme Court of the United States

MICROSOFT CORPORATION,

*Petitioner,*

*v.*

AT&T CORP.,

*Respondent.*

**On Writ Of Certiorari**
**To The United States Court Of Appeals**
**For The Federal Circuit**

## SUPPLEMENTAL JOINT APPENDIX

THEODORE B. OLSON
*Counsel of Record*
MIGUEL A. ESTRADA
MARK A. PERRY
MATTHEW D. MCGILL
AMIR C. TAYRANI
GIBSON, DUNN & CRUTCHER LLP
1050 Connecticut Ave., N.W.
Washington, D.C. 20036
(202) 955-8500

*Counsel for Petitioner*

SETH P. WAXMAN
*Counsel of Record*
JONATHAN E. NUECHTERLEIN
MARK C. FLEMING
WILMER CUTLER PICKERING
HALE AND DORR LLP
1875 Pennsylvania Ave., N.W.
Washington, D.C. 20006
(202) 663-6000

*Counsel for Respondent*

PETITION FOR WRIT OF CERTIORARI FILED FEBRUARY 17, 2006
CERTIORARI GRANTED OCTOBER 27, 2006

## TABLE OF CONTENTS

**Page**

United States Patent RE32,580 ..... 1

United States Patent 6,725,262 ..... 20

United States Patent 6,727,917 ..... 40

United States Patent 6,738,773 ..... 53

# United States Patent [19]

**Atal et al.**

[11] E **Patent Number: Re. 32,580**

[45] **Reissued Date of Patent: Jan. 19, 1988**


[54] **DIGITAL SPEECH CODER**


[75] Inventors: **Bishnu S. Atal, New Providence; Joel R. Remde, Elizabeth, both of N.J.**

[73] Assignee: **American Telephone and Telegraph Company, AT&T Bell Laboratories, Murray Hill, N.J.**



[21] Appl. No.: **909,319**

[22] Filed: **Sep. 18, 1986**

**Related U.S. Patent Documents**

Reissue of:

[64] Patent No.: **4,472,832**
Issued: **Sep. 18, 1984**
Appl. No.: **326,371**
Filed: **Dec. 1, 1981**

[51] Int. Cl.[4] ..... **G10L 5/00**

[52] U.S. Cl. ..... **381/40**

[58] Field of Search ..... 381/29–40, 381/49; 364/513.5



[56] **References Cited**

U.S. PATENT DOCUMENTS

| | | | |
|---|---|---|---|
| 3,624,302 | 11/1971 | Atal | 364/513.5 |
| 3,740,476 | 6/1973 | Atal | 381/49 |
| 4,130,729 | 12/1978 | Gagnon | 381/30 |
| 4,133,976 | 1/1979 | Atal et al. | 364/513.5 X |
| 4,140,876 | 2/1979 | Gagnon | 381/33 |
| 4,184,049 | 1/1980 | Crochier et al. | 381/37 |



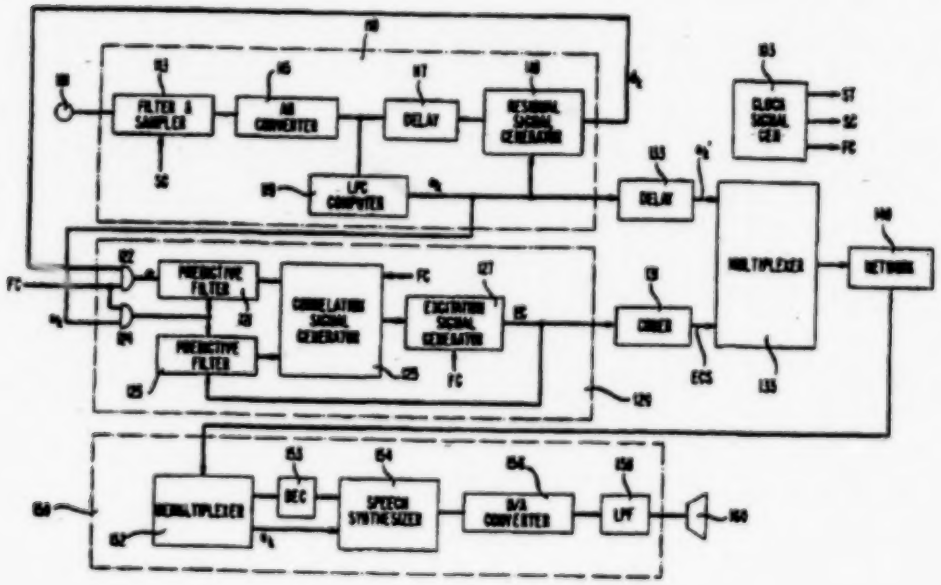
*Primary Examiner*—E. S. Matt Kemeny
*Attorney, Agent, or Firm*—Jack S. Cubert



[57] **ABSTRACT**

An improved speech analysis and synthesis system wherein LPC parameters and a modified residual signal for excitation is transmitted: the excitation signal is the cross correlation of the residual signal and the LPC-recreated original signal.


**43 Claims, 7 Drawing Figures**



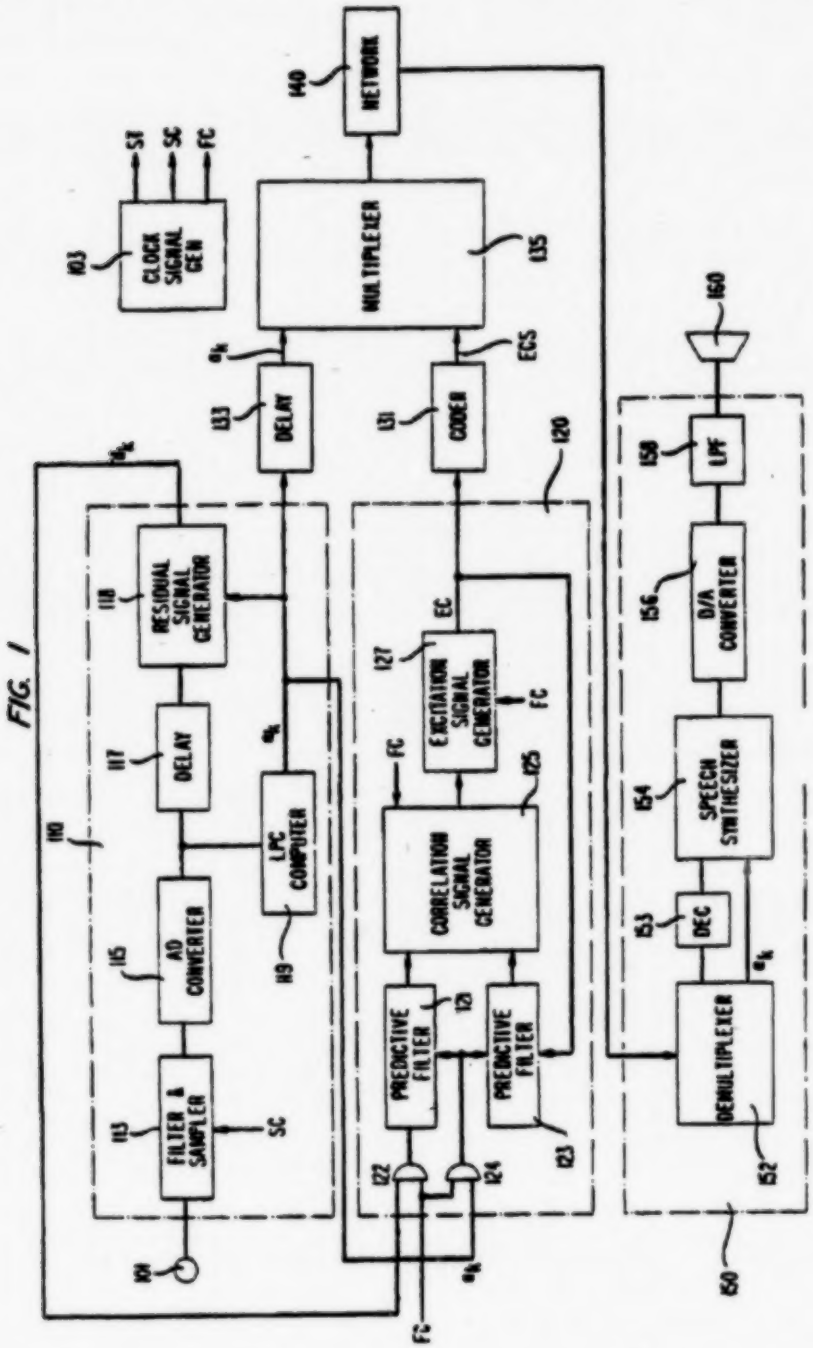

NETWORK
140
MULTIPLEXER
135
CLOCK SIGNAL GEN
103
ST
SC
FC
DELAY
133
CODER
131
ECS
120
RESIDUAL SIGNAL GENERATOR
118
DELAY
117
AD CONVERTER
115
FILTER & SAMPLER
113
SC
LPC COMPUTER
119
110
101
EXCITATION SIGNAL GENERATOR
127
EC
FC
CORRELATION SIGNAL GENERATOR
125
PREDICTIVE FILTER
121
PREDICTIVE FILTER
123
122
124
LPF
158
D/A CONVERTER
156
SPEECH SYNTHESIZER
154
DEC
153
DEMULTIPLEXER
152
150
160

FIG. 1

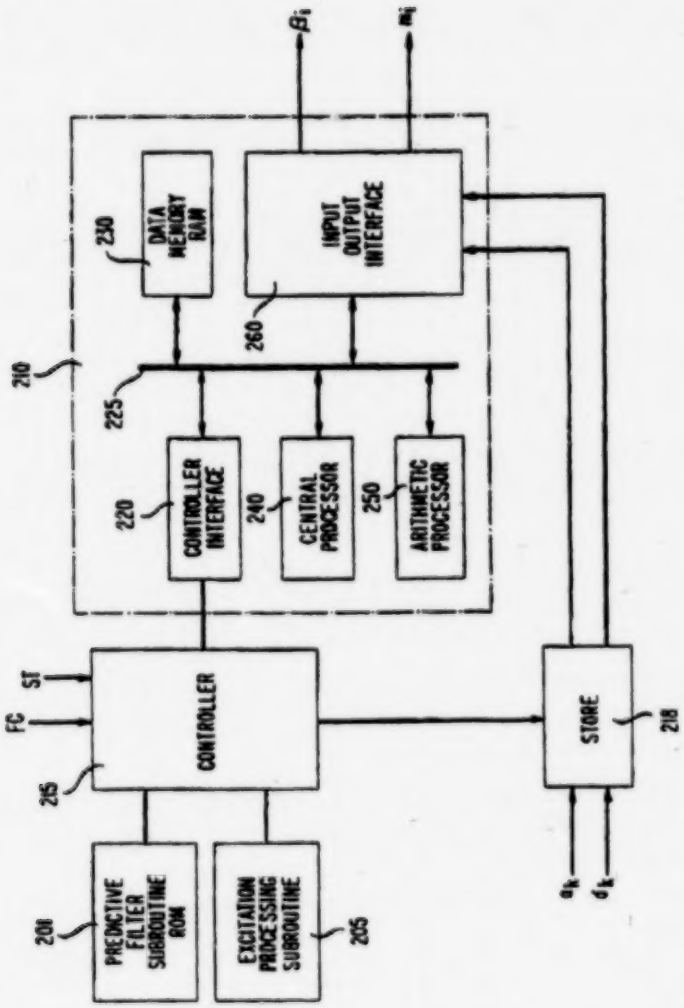

DATA MEMORY RAM
230
INPUT OUTPUT INTERFACE
260
210
225
CONTROLLER INTERFACE
220
CENTRAL PROCESSOR
240
ARITHMETIC PROCESSOR
250
CONTROLLER
215
ST
FC
STORE
218
PREDICTIVE FILTER SUBROUTINE ROM
201
EXCITATION PROCESSING SUBROUTINE
205

FIG. 2

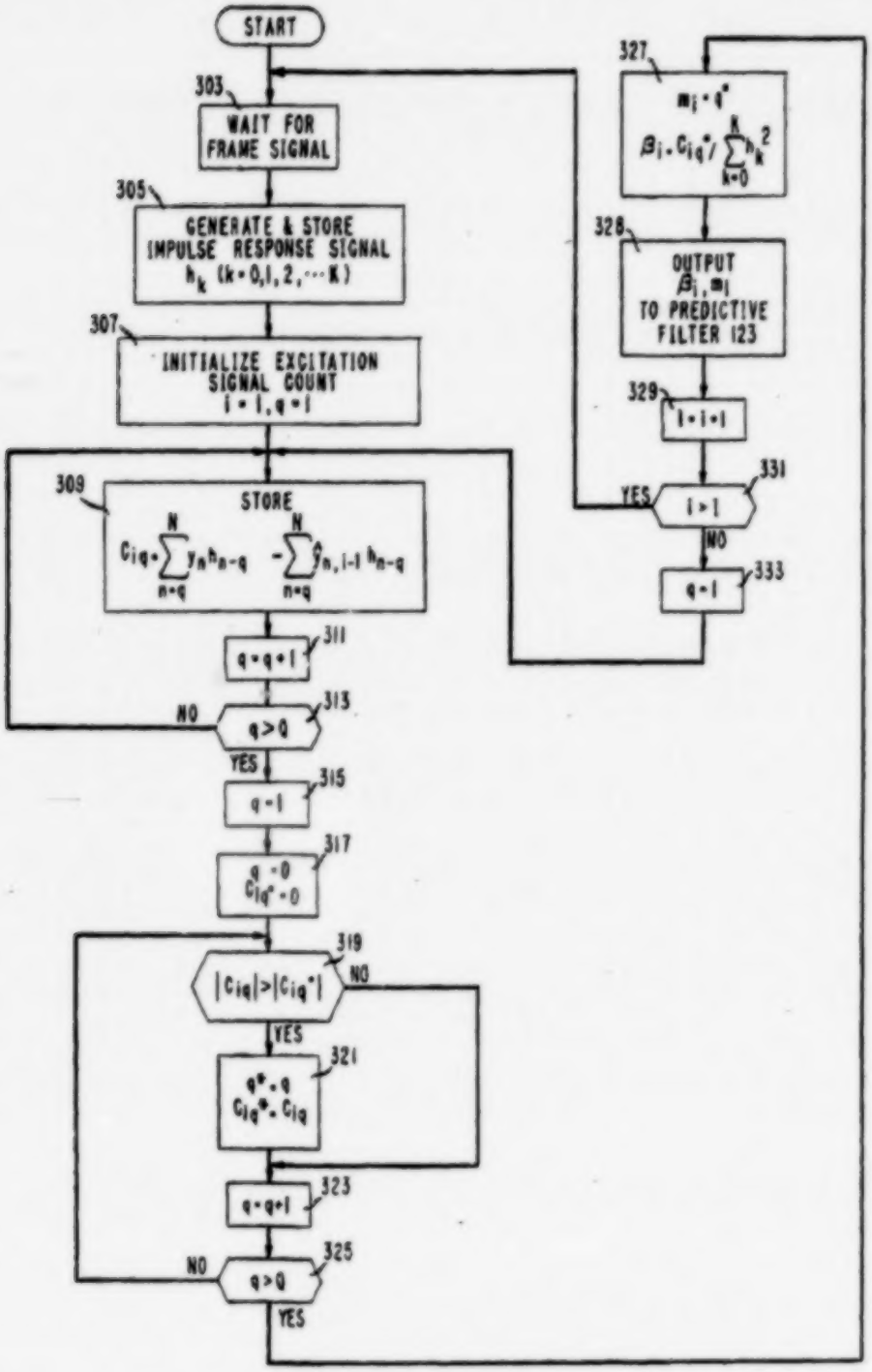

START
303
WAIT FOR FRAME SIGNAL
305
GENERATE & STORE IMPULSE RESPONSE SIGNAL $h_k$ (k=0,1,2,···K)
307
INITIALIZE EXCITATION SIGNAL COUNT i=1, q=1
309
STORE $C_{iq}=\sum_{n=q}^{N} y_n h_{n-q} - \sum_{n=q}^{N} \hat{y}_{n,i-1} h_{n-q}$
311
q=q+1
313
q>Q
NO
YES
315
q=1
317
q=0 $C_{iq}^*=0$
319
$|C_{iq}|>|C_{iq}^*|$
NO
YES
321
$q^*=q$ $C_{iq}^*=C_{iq}$
323
q=q+1
325
q>Q
NO
YES
327
$m_i=q^*$ $\beta_i=C_{iq}^*/\sum_{k=0}^{K} h_k^2$
328
OUTPUT $\beta_i, m_i$ TO PREDICTIVE FILTER 123
329
i=i+1
331
i>I
YES
NO
333
q=1

FIG. 4

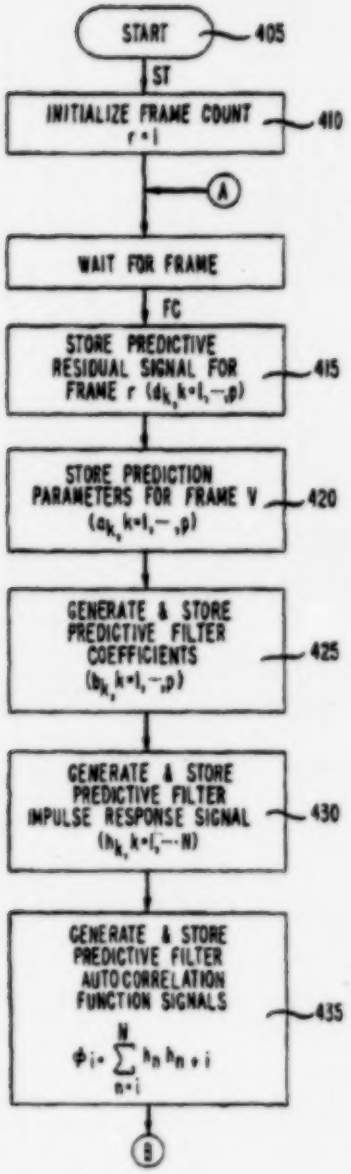
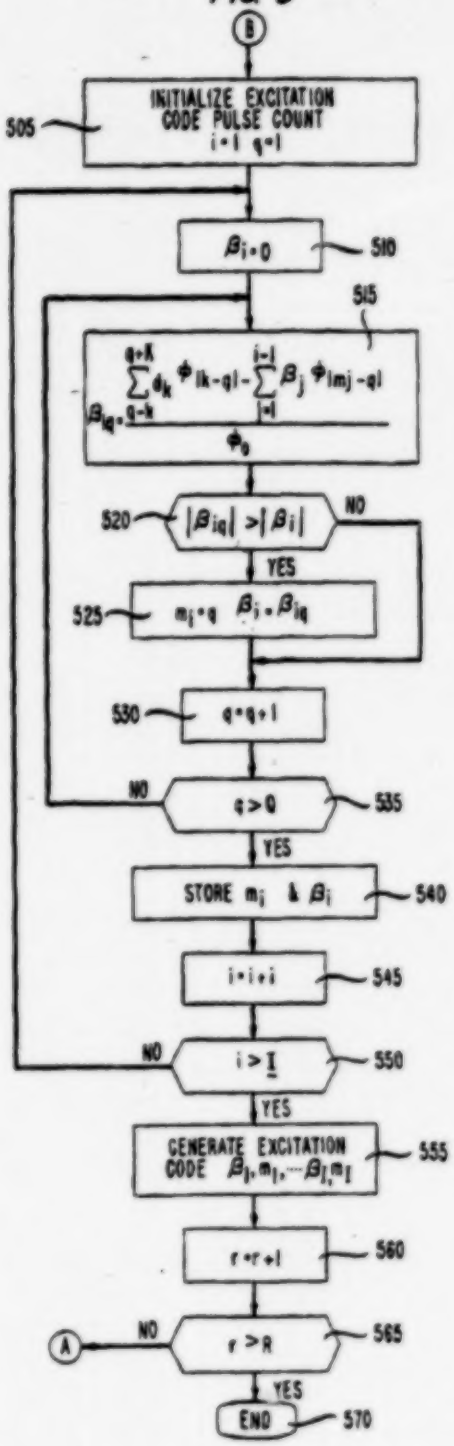

START
405
ST
INITIALIZE FRAME COUNT
r=1
410
A
WAIT FOR FRAME
FC
STORE PREDICTIVE RESIDUAL SIGNAL FOR FRAME r ($d_k$, k=1,—,p)
415
STORE PREDICTION PARAMETERS FOR FRAME V ($a_k$, k=1,—,p)
420
GENERATE & STORE PREDICTIVE FILTER COEFFICIENTS ($b_k$, k=1,—,p)
425
GENERATE & STORE PREDICTIVE FILTER IMPULSE RESPONSE SIGNAL ($h_k$, k=1,—N)
430
GENERATE & STORE PREDICTIVE FILTER AUTOCORRELATION FUNCTION SIGNALS
$\phi_i = \sum_{n=i}^{N} h_n h_{n-i}$
435
B

FIG. 5


B
INITIALIZE EXCITATION CODE PULSE COUNT
i=1 q=1
505
$\beta_i = 0$
510
515
$\beta_{iq} = \frac{\sum_{q-k}^{q+k} d_k \phi_{|k-q|} - \sum_{i=1}^{i-1} \beta_j \phi_{|mj-q|}}{\phi_0}$
520
$|\beta_{iq}| > |\beta_i|$
NO
YES
525
$m_i = q$ $\beta_i = \beta_{iq}$
530
q=q+1
NO
q>0
535
YES
STORE $m_i$ & $\beta_i$
540
i=i+1
545
NO
i>I
550
YES
GENERATE EXCITATION CODE $\beta_1, m_1, \cdots \beta_I, m_I$
555
r=r+1
560
A
NO
r>R
565
YES
END
570

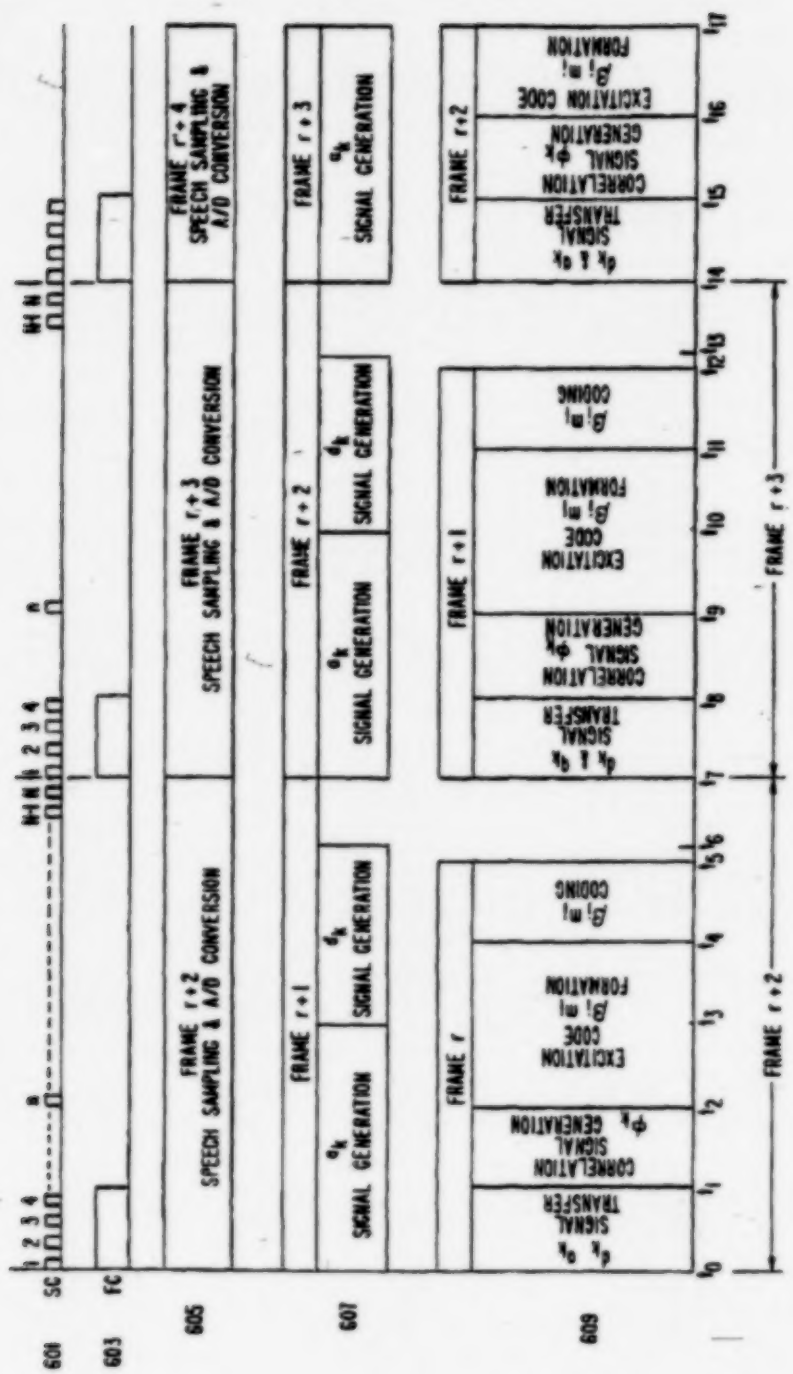
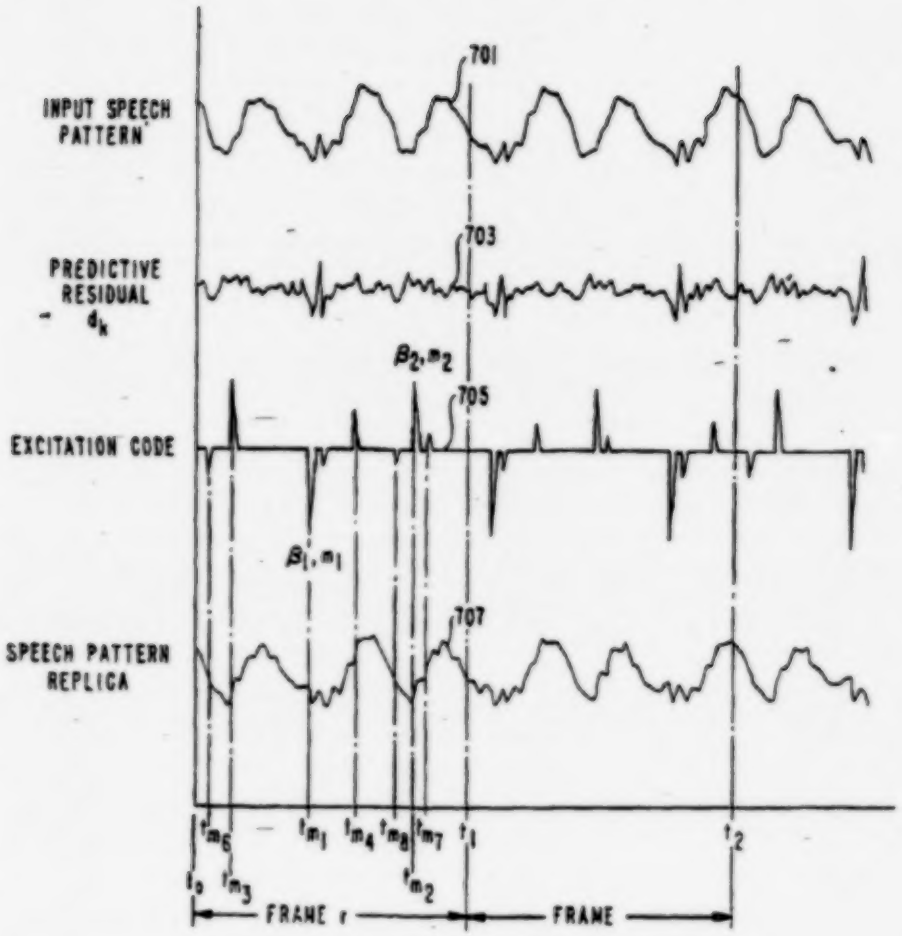

601
SC
1 2 3 4
n
603
FC
605
FRAME r+2
SPEECH SAMPLING & A/D CONVERSION
FRAME r+3
SPEECH SAMPLING & A/D CONVERSION
FRAME r+4
SPEECH SAMPLING & A/D CONVERSION
607
FRAME r+1
SIGNAL GENERATION
FRAME r+2
SIGNAL GENERATION
FRAME r+3
SIGNAL GENERATION
609
FRAME r
SIGNAL TRANSFER
CORRELATION SIGNAL GENERATION
EXCITATION CODE FORMATION
CODING
FRAME r+1
SIGNAL TRANSFER
CORRELATION SIGNAL GENERATION
EXCITATION CODE FORMATION
CODING
FRAME r+2
SIGNAL TRANSFER
CORRELATION SIGNAL GENERATION
EXCITATION CODE FORMATION
FRAME r+2
FRAME r+3

FIG. 6

FIG. 7


INPUT SPEECH PATTERN
701
PREDICTIVE RESIDUAL $d_k$
703
$\beta_2, m_2$
705
EXCITATION CODE
$\beta_1, m_1$
SPEECH PATTERN REPLICA
707
$t_{m_6}$
$t_{m_1}$
$t_{m_4}$
$t_{m_8}$
$t_{m_7}$
$t_1$
$t_2$
$t_0$
$t_{m_3}$
$t_{m_2}$
FRAME
FRAME

# DIGITAL SPEECH CODER

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue.

Our invention relates to speech processing and more particularly to digital speech coding arrangements.

Digital speech communication systems including voice storage and voice response facilities utilize signal compression to reduce the bit rate needed for storage and/or transmission. As is well known in the art, a speech pattern contains redundancies that are not essential to its apparent quality. Removal of redundant components of the speech pattern significantly lowers the number of digital codes required to construct a replica of the speech. The subjective quality of the speech replica, however, is dependent on the compression and coding techniques.

One well known digital speech coding system such as disclosed in U.S. Pat. No. 3,624,302 issued Nov. 30, 1971 includes linear prediction analysis of an input speech signal. The speech signal is partitioned into successive intervals and a set of parameters representative of the interval speech is generated. The parameter set includes linear prediction coefficient signals representative of the spectral envelope of the speech in the interval, and pitch and voicing signals corresponding to the speech excitation. These parameter signals may be encoded at a much lower bit rate than the speech signal waveform itself. A replica of the input speech signal is formed from the parameter signal codes by synthesis. The synthesizer arrangement generally comprises a model of the vocal tract in which the excitation pulses are modified by the spectral envelope representative prediction coefficients in an all pole predictive filter.

The foregoing pitch excited linear predictive coding is very efficient. The produced speech replica, however, exhibits a synthetic quality that is often difficult to understand. In general, the low speech quality results from the lack of correspondence between the speech pattern and the linear prediction model used. Errors in the pitch code or errors in determining whether a speech interval is voiced or unvoiced cause the speech replica to sound disturbed or unnatural. Similar problems are also evident in formant coding of speech. Alternative coding arrangements in which the speech excitation is obtained from the residual after prediction, e.g., ADPCM or APC, provide a marked improvement because the excitation is not dependent upon an inexact model. The excitation bit rate of these systems, however, is at least an order of magnitude higher than the linear predictive model. Attempts to lower the excitation bit rate in the residual type systems have generally resulted in a substantial loss in quality. It is an object of the invention to provide improved speech coding of high quality at lower bit rates than residual coding schemes.

## BRIEF SUMMARY OF THE INVENTION

We have found that the foregoing residual encoding problems may be solved by forming a pattern predictive of a pattern (e.g. speech pattern) to be encoded and comparing the pattern to be encoded with the predictive pattern on a frame by frame basis. The differences between the pattern to be encoded and the predictive pattern over each frame are utilized to form a coded signal of a prescribed format which coded signal modifies the predictive pattern to minimize the frame differences. The bit rate of the prescribed format coded signal is selected so that the modified predictive pattern approximates the speech pattern to a desired level consistent with coding requirements.



The invention is directed to a sequential pattern processing arrangement in which the sequential pattern is partitioned into successive time intervals. In each time interval, a set of signals representative of the interval sequential pattern and a signal representative of the differences between the interval sequential pattern and the interval representative signal set are generated. A first signal corresponding to the interval pattern is formed responsive to said interval pattern representative signals and said interval differences representative signal and a second interval corresponding signal is generated responsive to said interval pattern representative signals. A signal corresponding to the differences between the first and second interval corresponding signals is formed and a third signal is produced responsive to said interval differences corresponding signal that alters the second signal to reduce the differences between said first and second interval corresponding signals.

According to one aspect of the invention, a speech pattern is partitioned into successive time intervals. In each interval, a set of signals representative of the speech pattern in each time interval and a signal representative of the differences between said interval speech pattern and the interval speech pattern representative signal set are generated. A first signal corresponding to the interval speech pattern is formed responsive to said interval speech representative signals and differences representative signal and a second interval corresponding signal is generated responsive to the interval speech pattern representative signals. A signal corresponding to the differences between the first and second interval representative signals is formed and a third signal is produced responsive to the interval differences corresponding signal that alters said second interval corresponding signal to reduce the differences corresponding signal.

According to another aspect of the invention, the third signal is utilized to construct a replica of the interval pattern.

In an embodiment of the invention, a set of predictive parameter signals is generated for each time frame from a speech signal. A prediction residual signal is formed responsive to the time frame speech signal and the time frame predictive parameters. The prediction residual signal is passed through a first predictive filter to produce a first speech representative signal for the time frame. An second speech representative signal is generated for the time frame in a second predictive filter from the frame prediction parameters. Responsive to the first speech representative and second speech representative signals of the time frame, a coded excitation signal is formed and applied to the second predictive filter to minimize the perceptually weighted mean squared difference between the frame first and second speech representative signals. The coded excitation signal and the predictive parameter signals are utilized to construct a replica of the time frame speech pattern.

## DESCRIPTION OF THE DRAWING

FIG. 1 depicts a block diagram of a speech processor circuit illustrative of the invention;

FIG. 2 depicts a block diagram of an excitation signal forming processor that may be used in the circuit of FIG. 1;

FIG. 3 shows a flow chart that illustrates the operation of the excitation signal forming circuit of FIG. 1;

FIGS. 4 and 5 show flow charts that illustrate the operation of the circuit of FIG. 2;

FIG. 6 shows a timing diagram that is illustrative of the operation of the excitation signal forming circuit of FIG. 1 and of FIG. 2; and

FIG. 7 shows waveforms illustrating the speech processing of the invention.

## DETAILED DESCRIPTION

FIG. 1 shows a general block diagram of a speech processor illustrative of the invention. In FIG. 1, a speech pattern such as a spoken message is received by microphone transducer 101. The corresponding analog speech signal therefrom is bandlimited and converted into a sequence of pulse samples in filter and sampler circuit 113 of prediction analyzer 110. The filtering may be arranged to remove frequency components of the speech signal above 4.0 KHz and the sampling may be at an 8.0 KHz rate as is well known in the art. The timing of the samples is controlled by sample clock CL from clock generator 103. Each sample from circuit 113 is transformed into an amplitude representative digital code in analog-to-digital converter 115.

The sequence of speech samples is supplied to predictive parameter computer 119 which is operative, as is well known in the art, to partition the speech signals into 10 to 20 ms intervals and to generate a set of linear prediction coefficient signals $a_k, k = 1,2, \ldots, p$ representative of the predicted short time spectrum of the $N>>p$ speech samples of each interval. The speech samples from A/D converter 115 are delayed in delay 117 to allow time for the formation of signals $a_k$. The delayed samples are supplied to the input of prediction residual generator 118. The prediction residual generator, as is well known in the art, is responsive to the delayed speech samples and the prediction parameters $a_k$ to form a signal corresponding to the difference therebetween. The formation of the predictive parameters and the prediction residual signal for each frame shown in predictive analyzer 110 may be performed according to the arrangement disclosed in U.S. Pat. No. 3,740,476 issued to B. S. Atal June 19, 1973 and assigned to the same assignee or in other arrangements well known in the art.

While the predictive parameter signals $a_k$ form an efficient representation of the short time speech spectrum, the residual signal generally varies widely from interval to interval and exhibits a high bit rate that is unsuitable for many applications. In the pitch excited vocoder, only the peaks of the residual are transmitted as pitch pulse codes. The resulting quality, however, is generally poor. Waveform 701 of FIG. 7 illustrates a typical speech pattern over two time frames. Waveform 703 shows the predictive residual signal derived from the pattern of waveform 701 and the predictive parameters of the frames. As is readily seen, waveform 703 is relatively complex so that encoding pitch pulses corresponding to peaks therein does not provide an adequate approximation of the predictive residual. In accordance with the invention, excitation code processor 130 receives the residual signal $d_k$ and the prediction parameters $a_k$ of the frame and generates an interval excitation code which has a predetermined number of bit positions. The resulting excitation code shown in waveform 705 exhibits a relatively low bit rate that is constant. A replica of the speech pattern of waveform 701 constructed from the excitation code and the prediction parameters of the frames is shown in waveform 707. As seen by a comparison of waveforms 701 and 707, higher quality speech characteristic of adaptive predictive coding is obtained at much lower bit rates.

The prediction residual signal $d_k$ and the predictive parameter signals $a_k$ for each successive frame are applied from circuit 110 to excitation signal forming circuit 120 at the beginning of the succeeding frame. Circuit 120 is operative to produce a multielement frame excitation code EC having a predetermined number of bit positions for each frame. Each excitation code corresponds to a sequence of $1 \leq l \leq I$ pulses representative of the excitation function of the frame. The amplitude $\beta_l$ and location $m_l$ of each pulse within the frame is determined in the excitation signal forming circuit so as to permit construction of a replica of the frame speech signal from the excitation signal and the predictive parameter signals of the frame. The $\beta_l$ and $m_l$ signals are encoded in coder 131 and multiplexed with the prediction parameter signals of the frame in multiplexer 135 to provide a digital signal corresponding to the frame speech pattern.

In excitation signal forming circuit 120, the predictive residual signal $d_k$ and the predictive parameter signals $a_k$ of a frame are supplied to filter 121 via gates 122 and 124, respectively. At the beginning of each frame, frame clock signal FC opens gates 122 and 124 whereby the $d_k$ signals are supplied to filter 121 and the $a_k$ signals are applied to filters 121 and 123. Filter 121 is adapted to modify signal $d_k$ so that the quantizing spectrum of the error signal is concentrated in the formant regions thereof. As disclosed in U.S. Pat. No. 4,133,976 issued to B. S. Atal et al, Jan. 9, 1979 and assigned to the same assignee, this filter arrangement is effective to mask the error in the high signal energy portions of the spectrum.

The transfer function of filter 121 is expressed in z transform notation as

$$H(z) = \frac{1}{1 - B(z)} \tag{1}$$

where $B(z)$ is controlled by the frame predictive parameters $a_k$.

Predictive filter 123 receives the frame predictive parameter signals from computer 119 and an artificial excitation signal EC from excitation signal processor 127. Filter 123 has the transfer function of Equation 1. Filter 121 forms a weighted frame speech signal y responsive to the predictive residual $d_k$ while filter 123 generates a weighted artificial speech signal $\hat{y}$ responsive to the excitation signal from signal processor 127. Signals y and $\hat{y}$ are correlated in correlation processor 125 which generates a signal E corresponding to the weighted difference therebetween. Signal E is applied to signal processor 127 to adjust the excitation signal EC so that the differences between the weighted speech representative signal from filter 121 and the weighted artificial speech representative signal from filter 123 are reduced.

The excitation signal is a sequence of I (I≦15) pulses. Each pulse has an amplitude $\beta_i$ and a location $m_i$. Processor 127 is adapted to successively form the $\beta_i$, $m_i$ signals which reduce the differences between the weighted frame speech representative signal from filter 121 and the weighted frame artificial speech representative signal from filter 123. The weighted frame speech representative signal may be expressed as:

$$y_n = \sum_{k=n-k}^{n} d_k h_{n-k} \quad 1 \leq n \leq N \tag{2}$$

and the weighted artificial speech representative signal of the frame may be expressed as

$$\hat{y}_n = \sum_{j=1}^{I} \beta_j h_{n-m_j} \quad 1 \leq n \leq N \tag{3}$$

where $h_n$ is the impulse response of filter 121 or filter 123.

The excitation signal formed in circuit 120 is a coded signal having elements $\beta_i$, $m_i$, $i=1,2,\ldots,I$. Each element represents a pulse in the time frame. $\beta_i$ is the amplitude of the pulse and $m_i$ is the location of the pulse in the frame. Correlation signal generator circuit 125 is operative to successively generate a correlation signal for each element. Each element may be located at time $1 \leq q \leq Q$ in the time frame. Consequently, the correlation processor circuit forms Q possible candidates for element i in accordance with Equation 4:

$$C_{iq} = \sum_{n=q}^{N} Y_n h_{n-q} - \sum_{n=q}^{N} \hat{y}_{n,i-1} h_{n-q} \tag{4}$$

where

$$\hat{y}_{n,i-1} = \sum_{j=1}^{i-1} \beta_j h_{n-m_j} \tag{5}$$

Excitation signal generator 127 receives the $C_{iq}$ signals from the correlation signal generator circuit, selects the $C_{iq}$ signal having the maximum absolute value and forms the $i_{th}$ element of the coded signal

$$\beta_i = C_{iq^*} / \sum_{k=0}^{K} h_k^2 \tag{6}$$

$$m_i = q^*$$

where $q^*$ is the location of the correlation signal having the maximum absolute value. The index i is incremented to i+1 and signal $\hat{y}_n$ at the output of predictive filter 123 is modified. The process in accordance with Equations 4, 5 and 6 is repeated to form element $\beta_{i+1}$, $m_{i+1}$. After the formation of element $\beta_I$, $m_I$, the signal having elements $\beta_1 m_1$, $\beta_2 m_2$, ..., $\beta_I m_I$ is transferred to coder 131. As is well known in the art, coder 131 is operative to quantize the $\beta_i m_i$ elements and to form a coded signal suitable for transmission to network 140.

Each of filters 121 and 123 in FIG. 1 may comprise a transversal filter of the type described in aforementioned U.S. Pat. No. 4,133,976. Each of processors 125 and 127 may comprise one of the processor arrangements well known in the art adapted to perform the processing required by Equations 4 and 6 such as the C.S.P., Inc. Macro Arithmetic Processor System 100 or other processor arrangements well known in the art. Processor 125 includes a read-only memory which permanently stores programmed instructions to control the $C_{iq}$ signal formation in accordance with Equation 4 and processor 127 includes a read-only memory which permanently stores programmed instructions to select the $B_i$, $m_i$ signal elements according to Equation 6 as is well known in the art. The program instructions in processor 125 are set forth in FORTRAN language form in Appendix A and the program instructions in processor 127 are listed in FORTRAN language form in Appendix B.

FIG. 3 depicts a flow chart showing the operation of processors 125 and 127 for each time frame. Referring to FIG. 3, the $h_k$ impulse response signals are generated in box 305 responsive to the frame predictive parameters for the transfer function of Equation 1. This occurs after receipt of the FC signal from clock 103 in FIG. 1 as per wait box 303. The element index i and the excitation pulse location index q are initially set to 1 in box 307. Upon receipt of signals $y_n$ and $\hat{y}_{n,i-1}$ from predictive filters 121 and 123, signal $C_{iq}$ is formed as per box 309. The location index q is incremented in box 311 and the formation of the next location $C_{iq}$ signal is initiated.

After the $C_{iQ}$ signal is formed for excitation signal element i in processor 125, processor 127 is activated. The q index in processor 127 is initially set to 1 in box 315 and the i index as well as the $C_{iq}$ signals formed in processor 125 are transferred to processor 127. Signal $C_{iq^*}$ which represents the $C_{iq}$ signal having the maximum absolute value and its location $q^*$ are set to zero in box 317. The absolute values of the $C_{iq}$ signals are compared to signal $C_{iq^*}$ and the maximum of these absolute values is stored as signal $C_{iq^*}$ in the loop including boxes 319, 321, 323, and 325.

After the $C_{iQ}$ signal from processor 125 has been processed, box 327 is entered from box 325. The excitation code element location $m_i$ is set to $q^*$ and the magnitude of the excitation code element $\beta_i$ is generated in accordance with Equation 6. The $\beta_i m_i$ element is output to predictive filter 123 as per box 328 and index i is incremented as per box 329. Upon formulation of the $\beta_I m_I$ element of the frame, wait box 303 is reentered from decision box 331. Processors 125 and 127 are then placed in wait states until the FC frame clock pulse of the next frame.

The excitation code in processor 127 is also supplied to coder 131. The coder is operative to transform the excitation code from processor 127 into a form suitable for use in network 140. The prediction parameter signals $a_k$ for the frame are supplied to an input of multiplexer 135 via delay 133 as prediction signals $a'_k$. The excitation coded signal ECS from coder 131 is applied to the other input of the multiplexer. The multiplexed excitation and predictive parameter codes for the frame are then sent to network 140.

Network 140 may be a communication system, the message store of a voice storage arrangement, or apparatus adapted to store a complete message or vocabulary of prescribed message units, e.g., words, phonemes, etc., for use in speech synthesizers. Whatever the message unit, the resulting sequence of frame codes from circuit 120 are forwarded via network 140 to speech synthesizer 150. The synthesizer, in turn, utilizes the frame excitation codes from circuit 120 as well as the frame predictive parameter codes to construct a replica of the speech pattern.

Demultiplexer 152 in synthesizer 150 separates the excitation code EC of a frame from the prediction parameters $a_k$ thereof. The excitation code, after being

decoded into an excitation pulse sequence in decoder 153, is applied to the excitation input of speech synthesizer filter 154. The $a_k$ codes are supplied to the parameter inputs of filter 154. Filter 154 is operative in response to the excitation and predictive parameter signals to form a coded replica of the frame speech signal as is well known in the art. D/A converter 156 is adapted to transform the coded replica into an analog signal which is passed through low-pass filter 158 and transformed into a speech pattern by transducer 160.

An alternative arrangement to perform the excitation code formation operations to circuit 120 may be based on the weighted means squared error between signals $y_n$ and $\hat{y}_n$. This weighted mean squared error upon forming $\beta_l$ and $m_l$ for the $l^{th}$ excitation signal pulse is

$$E_l = \sum_{n=1}^{N} \left( y_n - \sum_{j=1}^{l} \beta_j h_{n-m_j} \right)^2 \tag{7}$$

where $h_n$ is the $n^{th}$ sample of the impulse response of $H(z)$, $m_j$ is the location of the $j^{th}$ pulse in the excitation code signal, and $\beta_j$ is the magnitude of the $j^{th}$ pulse.

The pulse locations and amplitudes are generated sequentially. The $i^{th}$ element of the excitation is determined by minimizing $E_l$ in Equation 7. Equation 7 may be rewritten as

$$E_l = \sum_{n=1}^{N} \left[ \left( y_n - \sum_{j=1}^{l-1} \beta_j h_{n-m_j} \right)^2 + \beta_l^2 h_{n-m_l}^2 + -2\beta_l \left( y_n h_{n-m_l} - \sum_{j=1}^{l-1} \beta_j h_{n-m_j} h_{n-m_l} \right) \right] \tag{8}$$

so that the known excitation code elements preceding $\beta_l, m_l$ appear only in the first term.

As is well known, the value of $\beta_l$ which minimizes $E_l$ can be determined by differentiating Equation 8 with respect to $\beta_l$ and setting

$$\frac{\partial E_l}{\partial \beta_l} = 0 \tag{9}$$

Consequently, the optimum value of $\beta_l$ is

$$= \frac{\sum_{k=m_l-K}^{m_l+K} d_k \phi_{|k-m_l|} - \sum_{j=1}^{l-1} \beta_j \phi_{|m_j-m_l|}}{\phi_0} \tag{10}$$

where

$$\phi_k = \sum_{n=k}^{K} h_n h_{n-k} \quad 0 \le k \le K \tag{11}$$

are the autocorrelation coefficients of the predictive filter impulse response signal $h_k$.

$\beta_l$ in Equation 10 is a function of the pulse location and is determined for each possible value thereof. The maximum of the $|\beta_l|$ values over the possible pulse locations is then selected. After $\beta_l$ and $m_l$ values are obtained, $\beta_{l+1}$ $m_{l+1}$ values are generated by solving Equation 10 in similar fashion. The first term of Equation 10, i.e.,

$$\sum_{k=m_l-K}^{m_l+K} d_k \phi_{|k-m_l|}$$

corresponds to the speech representative signal of the frame at the output of predictive filter 121. The second term of Equation 10, i.e.,

$$\sum_{j=1}^{l-1} \beta_j \phi_{|m_j-m_l|}$$

corresponds to the artificial speech representative signal of the frame at the output of predictive filter 123. $\beta_l$ is the amplitude of an excitation pulse at location $m_l$ which minimizes the difference between the first and second terms.

The data processing circuit depicted in FIG. 2 provides an alternative arrangement to excitation signal forming circuit 120 of FIG. 1. The circuit of FIG. 2 yields the excitation code for each frame of the speech pattern in response to the frame prediction residual signal $d_k$ and the frame prediction parameter signals $a_k$ in accordance with Equation 10 and may comprise the previously mentioned C.S.P., Inc. Macro Arithmetic Processor System 100 or other processor arrangements well known in the art.

Referring to FIG. 2, processor 210 receives the predictive parameter signals $a_k$ and the prediction residual signals $d_n$ of each successive frame of the speech pattern from circuit 110 via store 218. The processor is operative to form the excitation code signal elements $\beta_1$ $m_1$, $\beta_2$, $m_2$, . . . , $\beta_l$, $m_l$ under control of permanently stored instructions in predictive filter subroutine read-only memory 201 and excitation processing subroutine read-only memory 205. The predictive filter subroutine of ROM 201 is set forth in Appendix C and the excitation processing subroutine in ROM 205 is set forth in Appendix D.

Processor 210 comprises common bus 225, data memory 230, central processor 240, arithmetic processor 250, controller interface 220 and input-output interface 260. As is well known in the art, central processor 240 is adapted to control the sequence of operations of the other units of processor 210 responsive to coded instructions from controller 215. Arithmetic processor 250 is adapted to perform the arithmetic processing on coded signals from data memory 230 responsive to control signals from central processor 240. Data memory 230 stores signals as directed by central processor 240 and provides such signals to arithmetic processor 250 and input-output interface 260. Controller interface 220 provides a communication link for the program instructions in ROM 201 and ROM 205 to central processor 240 via controller 215, and input-output interface 260 permits the $d_k$ and $a_k$ signal to be supplied to data memory 230 and supplies output signals $\beta_l$ and $m_l$ from the data memory to coder 131 in FIG. 1.

The operation of the circuit of FIG. 2 is illustrated in the filter parameter processing flow chart of FIG. 4, the excitation code processing flow chart of FIG. 5, and the timing chart of FIG. 6. At the start of the speech signal, box 410 in FIG. 4 is entered via box 405 and the frame count r is set to the first frame by a single pulse ST from clock generator 103. FIG. 6 illustrates the operation of the circuit of FIGS. 1 and 2 for two successive frames. Between times $t_0$ and $t_7$ in the first frame, prediction

analyzer 110 forms the speech pattern samples of frame r+2 as in waveform 605 under control of the sample clock pulses of waveform 601. Analyzer 110 generates the $a_k$ signals corresponding to frame r+1 between times $t_0$ and $t_3$ and forms predictive residual signal $d_k$ between times $t_3$ and $t_6$ as indicated in waveform 607. Signal FC (waveform 603) occurs between times $t_0$ and $t_1$. The signals $d_k$ from residual signal generator 118 previously stored in store 218 during the preceding frame are placed in data memory 230 via input-output interface 260 and common bus 225 under control of central processor 240. As indicated operation box 415 of FIG. 4, these operations are responsive to frame clock signal FC. The frame prediction parameter signals $a_k$ from prediction parameter computer 119 previously placed in store 218 during the preceding frame are also inserted in memory 230 as per operation box 420. These operations occur between times $t_0$ and $t_1$ on FIG. 6.

After insertion of the frame $d_k$ and $a_k$ signals into memory 230, box 425 is entered and the predictive filter coefficients $b_k$ corresponding to the transfer function of Equation 1:

$$b_k = \alpha^k a_k, k = 1, 2, \ldots, p \tag{12}$$

are generated in arithmetic processor 250 and placed in data memory 230. p is typically 16 and $\alpha$ is typically 0.85 for a sampling rate of 8 KHz. The predictive filter impulse response signals $h_k$

$$h_0 = 1 \tag{13}$$

$$h_k = \sum_{i=1}^{\min(k-1,p)} b_i h_{k-i}, k = 1, 2, \ldots, K$$

are then generated in arithmetic processor 250 and stored in data memory 230. When the $h_k$ impulse response signal is stored, box 435 is entered and the predictive filter autocorrelation signals of Equation 11 are generated and stored.

At time $t_2$ in FIG. 6, controller 215 disconnects ROM 201 from interface 220 and connects excitation processing subroutine ROM 205 to the interface. The formation of the $\beta_i$, $m_i$ excitation pulse codes shown in the flow chart of FIG. 5 is then initiated. Between times $t_2$ and $t_4$ in FIG. 6, the excitation pulse sequence is formed. Excitation pulse index i is initially set to 1 and pulse location index q is set to 1 in box 505. $\beta_1$ is set to zero in box 510 and operation box 515 is entered to determine $\beta_q = \beta_{11}$. $\beta_{11}$ is the optimum excitation pulse at location q=1 of the frame. The absolute value of $\beta_{11}$ is then compared to the previously stored $\beta_i$ in decision box 520. Since $\beta_1$ is initially zero, the $m_i$ code is set to q=1 and the $\beta_i$ code is set to $\beta_{11}$ in box 525.

Location index q is then incremented in box 530 and box 515 is entered via decision box 535 to generate signal $\beta_{12}$. The loop including boxes 515, 520, 525, 530 and 535 is iterated for all pulse location values $1 \leq q \leq Q$. After the $Q^{th}$ iteration, the first excitation pulse amplitude $\beta_1 = \beta_{1q^*}$ and its location in the frame $m_1 = q^*$ are stored in memory 230. In this manner, the first of the I excitation pulses is determined. Referring to waveform 705 in FIG. 7, frame r occurs between times $t_0$ and $t_1$. The excitation code for the frame consists of 8 pulses. The first pulse of amplitude $\beta_1$ and location $m_1$ occurs at time $t_{m1}$ in FIG. 7 as determined in the flow chart of FIG. 5 for index i=1.

Index i is incremented to the succeeding excitation pulse in box 545 and operation box 515 is entered via box 550 and box 510. Upon completion of each iteration of the loop between boxes 510 and 550, the excitation signal is modified to further reduce the signal of Equation 7. Upon completion of the second iteration, pulse $\beta_2 m_2$ (time $t_{m2}$ in waveform 705) is formed. Excitation pulses $\beta_3 m_3$ (time $t_{m3}$), $\beta_4 m_4$ (time $t_{m4}$), $\beta_5 m_5$ (time $t_{m5}$), $\beta_6 m_6$ (time $t_{m6}$), $\beta_7 m_7$ (time $t_{m7}$), and $\beta_8 m_8$ (time $t_{m8}$), are then successively formed as index i is incremented.

After the $I^{th}$ iteration (waveform 609 at $t_4$), box 555 is entered from decision box 550 and the current frame excitation code $\beta_1 m_1, \beta_2 m_2, \ldots, \beta_{Im} I$ is generated therein. The frame index is incremented in box 560 and the predictive filter operations of FIG. 4 for the next frame are started in box 415 at time $t_7$ in FIG. 6. Upon the occurrence of the FC clock signal for the next frame at $t_7$ in FIG. 6, the predictive parameter signals for frame r+3 are formed (waveform 605 between times $t_7$ and $t_{14}$), the $a_k$ and $d_k$ signals are generated for frame r+2 (waveform 607 between times $t_7$ and $t_{13}$), and the excitation code for frame r+1 is produced (waveform 609 between times $t_7$ and $t_{12}$).

The frame excitation code from the processor of FIG. 2 is supplied via input-output interface 260 to coder 131 in FIG. 1 as is well known in the art. Coder 131 is operative as previously mentioned in quantize and format the excitation code for application to network 140. The $a_k$ prediction parameter signals for the frame are applied to one input of multiplexer 135 through delay 133 so that the frame excitation code from coder 131 may be appropriately multiplexed therewith.

The invention has been described with reference to particular illustrative embodiments. It is apparent to those skilled in the art with various modifications may be made without departing from the scope and the spirit of the invention. For example, the embodiments described herein have utilized linear predictive parameters and a predictive residual. The linear predictive parameters may be replaced by formant parameters or other speech parameters well known in the art. The predictive filters are then arranged to be responsive to the speech parameters that are utilized and to the speech signal so that the excitation signal formed in circuit 120 of FIG. 1 is used in combination with the speech parameter signals to construct a replica of the speech pattern of the frame in accordance with the invention. The encoding arrangement of the invention may be extended to sequential patterns such as biological and geological patterns to obtain efficient representations thereof.

## APPENDIX A

```
C     THIS SUBROUTINE IMPLEMENTS BOX NOS 125-FIG. 1
C      ++++++ CORRELATION SIGNAL GENERATOR ++++++

      COMMON Y(110),YHAT(110),H(15),CI(110),A(15),F(16),
     &BETA(12),M(12)
```

```
                11                                          12
          INTEGER I,K,Q,QSTAR,QMAX
          DATA NLPC/16/,KMAX/15/,NMAX/110/,
         &QMAX/95/,ALPHA/0.85/,IMAX/12/
C++++     COMPUTE COEFFICIENTS FOR THE PREDICTIVE FILTER
         G=1
         DO101K=1,NLPC
         G=G*ALPHA
  101    F(K)=A(K)*G

C++++      BOX 305 FIG. 3
C++++      COMPUTE IMPULSE RESPONSE OF
C++++        PREDICTIVE FILTER H(Z)
C++++      H(0) IS STORED AS H(1) , H(1) IS
C++++        STORED AS H(2), AND SO ON
         H(1)=1
         DO102K=2,NLPC
         H(K)=0
         DO102I=1,K-1
  102    H(K)=H(K)+F(I)*H(K-I)

         DO103K=NLPC+1,KMAX
         H(K)=0
         DO103I=1,NLPC
  103    H(K)=H(K)+F(I)*H(K-I)

         SUMSQH=0
         DO104K=1,KMAX
  104    SUMSQH=SUMSQH+H(K)*H(K)

C++++    SET INITIAL EXCITATION SIGNAL COUNT - BOX 307
         I=1
         Q=1

  500    Q=1
  200    CONTINUE

C++++      COMPUTE CORRELATION SIGNAL - BOX 309
         DO201N=Q,NMAX
  201    CI(Q)=CI(Q)+(Y(N)-YHAT(N))*H(N-Q+1)
         Q=Q+1
         IF(Q.LE.QMAX)GOTO200

         CALL BOX127(I,SUMSQH)

         I=I+1
         IF(I.LE.IMAX)GOTO500
         RETURN
         END
```

APPENDIX B

```
C     THIS SUBROUTINE IMPLEMENTS BOX NOS 127 - FIG. 1
C       ++++++ EXCITATION SIGNAL GENERATOR  ++++++

      COMMON Y(110),YHAT(110),H(15),CI(110),A(16,F(16),
     &BETA(12),M(12)
```

```
      INTEGER I,K,Q,QSTAR,QMAX
      DATA NLPC/16/,KMAX/15/,NMAX/110/,QMAX/95/,
     &ALPHA/0.85/, IMAX/12/
C++++    FIND PEAK OF THE CORRELATION SIGNAL -
C++++      BOX 315-325
      Q=1
      QSTAR=0
      CIQSTAR=0

300   IF(ABS(CI(Q)).LT.ABS(CIQSTAR))GOTO301
      QSTAR=Q
      CIQSTAR=CI(Q)

301   Q=Q+1
      IF(Q.LE.QMAX)GOTO300

      M(I)=QSTAR
      BETA(I)=CIQSTAR/SUMSQH

      RETURN
      END
```

APPENDIX C

```
C     THIS SUBROUTINE IMPLEMENTS ROM 201 - FIG. 2
C      ++++++  PREDICTIVE FILTER ++++++

      COMMON D(110),H(15),BETAI(80),A(16),F(16),
     &PHI(15),BETA(12),M(12)
      INTEGER I,K,Q,QSTAR,QMAX
      DATA NLPC/16/,KMAX/15/,NMAX/110/,QMAX/80/,
     &ALPHA/0.85/, IMAX/12/

C++++    READ PREDICTIVE RESIDUAL SIGNAL - BOX 415
      CALL INPUT (D(29),80)

C++++    READ PREDICTION PARAMETERS - BOX 420
      CALL INPUT(A,16)

C++++    COMPUTE COEFFICIENTS FOR THE PREDICTIVE
C++++     FILTER - BOX 425
      G=1
      DO101K=1,NLPC
      G=G*ALPHA
  101 F(K)=A(K)*G

C++++    COMPUTE IMPULSE RESPONSE OF PREDICTIVE
C++++     FILTER H(Z)
C++++    H(0) IS STORED AS H(1) , H(1)
C++++     IS STORED AS H(2), AND SO ON
C++++    BOX 430
      H(1)=1
      DO102K=2,NLPC
      H(K)=0
      DO102I=1,K-1
  102 H(K)=H(K)+F(I)*H(K-I)
```

```
      DO103K=NLPC+1,KMAX
      H(K)=0
      DO103I=1,NLPC
  103 H(K)=H(K)+F(I)*H(K-I)
C++++   COMPUTE AUTOCORRELATION FUNCTION SIGNALS -
C++++    BOX 435
      DO104K=1,KMAX
      PHI(K)=0
      DO104N=K,KMAX
  104 PHI(K)=PHI(K)+H(N)*H(N+K-1)

      RETURN
      END
```

APPENDIX D

```
C      THIS SUBROUTINE IMPLEMENTS ROM 205 - FIG. 2
C         ++++++ EXCITATION PROCESSING ++++++

       COMMON D(110),H(15),BETAI(80),A(16),
      &PHI(15),BETA(12),M(12)
       INTEGER I,K,Q,QSTAR,QMAX
       DATA NLPC/16/,KMAX/15/,NMAX/110/,QMAX/80/,
      &ALPHA/0.85/, IMAX/12/

C++++  COMPUTE INITIAL BETAI SIGNAL (I=1)
C++++  TERM NO 1 EQUATION 10 AND BOX 515
       DO105Q=1,QMAX
       BETAI(Q)=0
       DO105N=Q,Q+2*KMAX-2
       K=N-KMAX+1
  105  BETAI(Q)=BETAI(Q)+D(N)*PHI(1+IABS(K-Q))

C++++   SET INITIAL EXCITATION SIGNAL COUNT - BOX 505
       I=1
       Q=1

  500  CONTINUE
       BETAMAX=0

  200  CONTINUE

C++++     COMPUTE BETAI SIGNAL - BOX 515
       IF(I.EQ.1)GOTO100
       DO201J=1,I-1
  201  BETAI(Q)=BETAI(Q)-BETA(J)*PHI(1+IABS(M(J)-Q))

C++++  FIND PEAK OF THE BETAI SIGNAL - BOX 520-525
  300  IF(ABS(BETAI(Q)).LT.BETAMAX)GOTO301

       M(I)=Q
       BETAMAX=ABS(BETAI(Q))
       BETA(I)=BETAI(Q)/PHI(I)

  301  Q=Q+1
       IF(Q.LE.QMAX)GOTO200
```

```
      I=I+1
      IF(I.LE.IMAX)GOTO500
CALL OUTPUT(SETA,IMAX)
CALL OUTPUT(M,IMAX)

DO5K=1,29
D(K)=D(K+80)

RETURN
END
```

What is claimed is:

1. A method for processing a sequential pattern comprising the steps of: partitioning said sequential pattern into successive time intervals; generating a set of signals representative of the sequential pattern of each time interval responsive to said time interval sequential pattern; generating a signal corresponding to the differences between said interval sequential pattern and the interval representative signal set responsive to said interval sequential pattern and said interval representative signals; forming a first signal corresponding to the interval pattern responsive to said interval pattern representative signals and said interval differences representative signal; generating a second interval corresponding signal responsive to said interval pattern representative signals; generating a signal corresponding to the differences between said first and second interval corresponding signals; producing a third signal responsive to said interval differences corresponding signal for altering said second signal to reduce the interval differences corresponding signal; and utilizing said third signal to construct a replica of said interval sequential pattern.

2. A method for processing a speech pattern comprising the steps of: partitioning the speech pattern into successive time intervals; generating a set of signals representative of said speech pattern of each time interval responsive to said interval speech pattern; generating a signal representative of the differences between said interval speech pattern and the interval speech pattern representative signal set responsive to said interval speech pattern and said interval speech pattern representative signals; forming a first signal corresponding to the interval speech pattern responsive to said interval speech pattern representative signals and the interval differences representative signal; forming a second interval corresponding signal responsive to the interval speech pattern representative signals; generating a signal corresponding to the differences between said first and second interval corresponding signals; and producing a third signal responsive to said interval differences corresponding signal for altering said second signal to reduce the interval differences corresponding signal.

3. A method for processing a speech pattern according to claim 2 wherein: said interval representative signal set generating step comprises generating a set of speech parameter signals representative of said interval speech pattern; said first interval corresponding signal forming step comprises generating said first interval corresponding signal responsive to said speech parameter signals and said differences representative signal; and said second interval corresponding signal forming step comprises generating said second interval corresponding signal responsive to said interval speech parameter signals.

4. A method for processing a speech pattern according to claim 3 wherein said speech parameter signal generating step comprises generating a set of signals representative of the interval speech spectrum.

5. A method for processing a speech pattern according to claim 4 wherein: said third signal producing step comprises generating a coded signal having at least one element responsive to the interval differences corresponding signal; and modifying said second interval corresponding signal responsive to said coded signal element.

6. A method for processing a speech pattern according to claim 5 wherein: said coded signal generating step comprises generating, for a predetermined number of times, a coded signal element responsive to said interval differences corresponding signal; and modifying said second interval corresponding signal responsive to said generated coded signal elements.

7. A method for processing a speech pattern according to claim 6 wherein: said differences corresponding signal generating step comprises generating a signal representative of the correlation between said first interval corresponding and second interval corresponding signals.

8. A method for processing a speech pattern according to claim 5 wherein said differences corresponding signal generating step comprises generating a signal representative of the mean squared difference between said first and second interval corresponding signals.

9. A method for processing a speech pattern according to claims 2, 3, or 4 further comprising the step of utilizing said third signal to construct a replica of said interval speech pattern.

10. A sequential pattern processor comprising means for partitioning a sequential pattern into successive time intervals; means responsive to each time interval sequential pattern for generating a set of signals representative of the sequential pattern of said time interval; means responsive to said interval sequential pattern and said interval representative signals for generating a signal representative of the differences between said interval sequential pattern and the interval representative signal set; means responsive to said interval pattern representative signals and said differences representative signal for forming a first signal corresponding to the interval pattern; means responsive to said interval pattern representative signals for generating a second interval corresponding signal; means for generating a signal corresponding to the differences between said first and second interval corresponding signals; and means responsive to said interval differences corresponding signal for producing a third signal for altering said second signal to reduce the interval differences

corresponding signal; and means for utilizing said third signal to construct a replica of said interval sequential pattern.

11. A speech processor comprising means for partitioning a speech pattern into successive time intervals; means responsive to each interval speech pattern for generating a set of signals representative of the speech pattern of said time interval; means responsive to said interval speech pattern and said interval speech pattern representative signals for generating a signal representative of the differences between said interval speech pattern and the interval representative signal set; means responsive to said speech interval signals and said interval differences representative signal for forming a first signal corresponding to the interval speech pattern; means responsive to said interval speech pattern representative signals for forming a second interval corresponding signal; means for generating a signal corresponding to the differences between said first and second interval corresponding signals; and means responsive to said interval differences corresponding signal for producing a third signal for altering said second interval corresponding signal to reduce the interval differences corresponding signal.

12. A speech processor according to claim 11 wherein: said speech interval representative signal set generating means comprises means for generating a set of signals representative of prescribed speech parameters of said interval speech pattern; said first interval corresponding signal forming means comprises means responsive to said interval prescribed speech parameter signals and said differences representative signal for generating said first interval corresponding signal; said second interval corresponding signal forming means comprises means responsive to said interval prescribed speech parameter signals for generating the second interval corresponding signal.

13. A speech processor according to claim 12 wherein said prescribed speech parameter signal generating means comprises means for generating a set of signals representative of the interval speech pattern spectrum.

14. A speech processor according to claim 13 wherein: said third signal producing means comprises means responsive to said interval differences corresponding signal for generating a coded signal having at least one element; and means responsive to said coded signal elements for modifying said second interval corresponding signal.

15. A speech processor according to claim 14 wherein: said coded signal generating means comprises means operative N times to produce an N element coded signal including means responsive to said differences corresponding signal for generating coded signal elements; and means responsive to the generated coded signal elements for modifying said second interval corresponding signal.

16. A speech processor according to claim 15 wherein: said interval differences corresponding signal generating means comprises means for generating a signal representative of the correlation between said first and second interval corresponding signals.

17. A speech processor according to claim 15 wherein said interval differences corresponding signal generating means comprises means for generating a signal representative of the mean squared difference between said first and second interval corresponding signals.

18. A speech processor according to claims 11, 12, or 13 further comprising the step of utilizing said third signal to construct a replica of said interval speech pattern.

19. A method for encoding a speech pattern comprising the steps of: partitioning a speech pattern into successive time frames; generating for each frame a set of speech parameter signals responsive to the frame speech pattern; generating a signal representative of the differences between the frame speech pattern and said speech parameter signal set responsive to said frame speech pattern and said frame speech parameter signals; generating a first signal corresponding to the frame speech pattern responsive to said frame speech parameter signals and said differences representative signal; generating a second frame corresponding signal responsive to said frame speech parameter signals; generating a signal corresponding to the differences between said first and second interval corresponding signals; and producing a coded signal responsive to said interval differences corresponding signal for modifying said second interval corresponding signal to reduce said interval differences corresponding signal.

20. A method for encoding a speech signal according to claim 19 further comprising combining said produced coded signal and said speech parameter signals to form a coded signal representative of the frame speech pattern.

21. A method for encoding a speech signal according to claim 19 wherein said speech parameter signal set generation comprises generating a set of linear predictive parameter signals for the frame responsive to said frame speech pattern; and said differences representative signal generation comprises generating a predictive residual signal responsive to said frame linear prediction parameter signals and said frame speech pattern.

22. A method for encoding a speech signal according to claim 21 wherein said coded signal producing step comprises generating a coded signal having at least one element responsive to said differences corresponding signal; and modifying said frame second signal responsive to said coded signal elements.

23. A method for encoding a speech pattern according to claim 21 wherein said signal producing step comprises generating a multielement coded signal by successively generating a coded signal element responsive to said differences corresponding signal and modifying said second signal responsive to said coded signal elements.

24. Apparatus for encoding a speech pattern comprising means for partitioning a speech pattern into successive time frames; means responsive to the frame speech pattern for generating for each frame a set of speech parameter signals; means responsive to said frame speech parameter signals and said frame speech pattern for generating a signal representative of the differences between said frame speech pattern and said frame speech parameter signal set; means responsive to said frame speech parameter signals and said differences representative signal for generating a first signal corresponding to said frame speech pattern; means responsive to said frame speech parameter signals for generating a second frame corresponding signal; means for generating a signal corresponding to the differences between said first and second frame corresponding signals; and means responsive to said frame differences corresponding signal for producing a third signal to

modify said second signal to reduce the frame differences corresponding signal.

25. Apparatus for encoding a speech pattern according to claim 24 further comprising means for combining said produced coded signal and said speech parameter signals to form a coded signal representative of the frame speech pattern.

26. Apparatus for encoding a speech pattern according to claim 24 wherein said speech parameter signal generating means comprises means responsive to said frame speech pattern for generating a set of linear predictive parameter signals for the frame; said differences representative signal generating means comprises means responsive to said frame linear prediction parameter signals and said frame speech pattern for generating a frame predictive residual signal; said first signal generating means comprises means responsive to said frame predictive parameter signals and said frame predictive residual signal for forming said first frame corresponding signal; and said second signal generating means comprises means responsive to said frame linear predictive parameter signals for forming said second frame corresponding signal.

27. Apparatus for encoding a speech pattern according to claim 26 wherein said coded signal producing means comprises means responsive to said difference corresponding signal for generating a coded signal having at least one element; and means responsive to said coded signal element for modifying said second signal.

28. Apparatus for encoding a speech pattern according to claim 26 wherein said coded signal producing means comprises means for generating a multielement coded signal including means operative successively for generating a coded signal element responsive to said differences corresponding signal and for modifying said second signal responsive to said coded signal elements.

29. A speech processor comprising means for partitioning a speech pattern into successive time frames; means responsive to the speech pattern of each frame for producing a set of predictive parameter signals and a predictive residual signal; means responsive to said frame predictive parameter and predictive residual signals for generating a first signal corresponding to the frame speech pattern; means responsive to said frame predictive parameter signals for generating a second frame corresponding signal; means responsive to said first and second frame corresponding signals for producing a signal corresponding to the differences between said first and second frame corresponding signals; means responsive to said frame differences corresponding signal for generating a coded excitation signal and for applying said coded excitation signal to said second signal generating means to reduce the differences corresponding signal.

30. A speech processor according to claim 29 further comprising means responsive to said frame coded excitation signal and said frame predictive parameter signals for constructing a replica of said frame speech pattern.

31. A speech processor according to claim 29 or claim 30 wherein said coded excitation signal generating means comprises means operative successively to form a multielement coded signal comprising means responsive to the differences corresponding signal for forming an element of said multielement code and for modifying said second signal responsive to said coded signal elements.

32. A method for processing a speech pattern according to claim 5, 6, 7, or 8 further comprising the step of utilizing said coded signal to construct a replica of said interval speech pattern.

33. A speech processor according to claim 14, 15, 16, or 17 further comprising means for utilizing said coded signal to construct a replica of said interval speech pattern.

34. A speech processor for producing a speech message comprising: means for receiving a sequence of speech message time interval signals, each speech interval signal including a plurality of spectral representative signals and an excitation representative signal for said time interval; means jointly responsive to said interval spectral representative signals and said interval excitation representative signal for generating a speech pattern corresponding to the speech message; said interval excitation speech signal being formed by the steps of: partitioning a speech message pattern into successive time intervals; generating a set of signals representative of said speech message pattern for each time interval responsive to said interval speech pattern; generating a signal representative of the differences between said interval speech pattern and said representative signal set responsive to said interval speech pattern and said interval respresentative signals; forming a first signal corresponding to the interval speech message pattern responsive to said speech message pattern interval representative signals and differences representative signals; forming a second interval corresponding signal responsive to said interval speech message pattern representative signals; generating a signal corresponding to the differences between said first and second interval corresponding signals; and producing a third signal responsive to said interval differences corresponding signal for altering said second interval corresponding signal to reduce the interval differences corresponding signal, said third signal being said interval excitation representative signal.

35. A speech processor according to claim 34 wherein said interval differences corresponding signal generating step comprises generating a signal representative of the correlation between said first interval corresponding signal and said second interval corresponding signal and said third signal producing step comprises forming a coded signal responsive to said correlation representative signal.

36. A speech processor according to claim 34 or 35 wherein said speech message interval spectral representative signals are time interval predictive parameter signals.

37. A method for producing a speech message comprising the steps of: receiving a sequence of speech message interval signals, each speech interval signal including a plurality of spectral representative signals and an excitation representative signal; and generating a speech pattern corresponding to the speech message jointly responsive to said interval spectral representative signals and said interval excitation representative signals; said interval excitation speech signal being formed by the steps of: partitioning a speech pattern into successive time intervals; generating a set of signals representative of the spectrum of said speech pattern for each time interval responsive to said interval speech pattern; generating a signal representative of the differences between said interval speech pattern and said interval speech pattern spectral representative signal set responsive to said interval speech pattern and said spectral representative signals; forming a first signal corresponding to the interval speech pattern responsive to

said interval spectral representative signals and said differences representative signal; forming a second interval corresponding signal responsive to said speech pattern interval spectral representative signals; generating a signal corresponding to the differences between said first and second interval corresponding signals; and producing a third signal responsive to said interval differences corresponding signal for altering said second interval corresponding signal to reduce the interval differences corresponding signal said third signal being said interval excitation signal.

38. A method for producing a speech message according to claim 37 wherein said interval differences corresponding signal generating step comprises generating a signal representative of the correlation between said first signal and said second signal and said third signal producing step comprises forming a prescribed format signal responsive to said correlation representative signal.

39. A method for producing a speech message according to claim 37 or 38 wherein said speech interval spectral representative signals are speech interval predictive parameter signals.

*40. Apparatus for producing a speech message comprising:*

*means for receiving a sequence of speech message signals for the successive time intervals of the speech message, each time interval speech message signal including a set of coded spectral representative signals for the time interval portion of said speech message and a plurality of pulse amplitude and location coded signals representative of the differences between the time interval portion of the speech message and the time interval portion of the speech message formed from said spectral representative signals;*

*means for converting the plurality of pulse amplitude and location codes of said time interval into a signal representative of the excitation of the time interval portion of said speech message; and*

*means jointly responsive to said interval spectral representative signals and said interval excitation representative signal for generating a speech pattern corresponding to the speech message of said time interval.*

*41. Apparatus for producing a speech message according to claim 40 wherein said converting means comprises means responsive to said amplitude and location codes for forming a sequence of pulses within said time interval representative of the excitation of the speech message portion of said time interval.*

*42. A method for producing a speech message comprising the steps of:*

*receiving a sequence of speech message signals for the successive time interval portions of the speech message, each time interval speech message signal including a set of coded spectral representative signals for the time interval portion of said speech message and a plurality of pulse amplitude and location coded signals representative of the differences between the time interval portion of the speech message and the time interval portion of the speech message formed from said spectral representative signals;*

*converting the plurality of pulse amplitude and location codes of said time interval into a signal representative of the excitation of the time interval portion of said speech message; and*

*generating a speech pattern corresponding to the speech message of said time interval jointly responsive to said interval spectral representative signals and said interval excitation representative signal.*

*43. A method for producing a speech message according to claim 42 wherein said converting step comprises forming a sequence of pulses within said time interval representative of the excitation of the speech message portion of said time interval responsive to said amplitude and location codes.*

* * * * *

US006725262B1

(12) **United States Patent**
**Choquier et al.**

(10) **Patent No.: US 6,725,262 B1**
(45) **Date of Patent: Apr. 20, 2004**

(54) **METHODS AND SYSTEMS FOR SYNCHRONIZING MULTIPLE COMPUTING DEVICES**

(75) Inventors: **Philippe Choquier**, Redmond, WA (US); **Martin J. Sleeman**, Seattle, WA (US); **William Russell More**, Bellevue, WA (US); **Alan G. Graham**, Redmond, WA (US); **Bradford W. Wright**, Bellevue, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/560,701**

(22) Filed: **Apr. 27, 2000**

(51) **Int. Cl.**[7] ..... **G06F 15/177**; G06F 15/16

(52) **U.S. Cl.** ..... 709/221; 709/217; 709/208

(58) **Field of Search** ..... 709/208, 220, 709/248, 217, 219, 221, 222; 707/201

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 5,790,677 A | | 8/1998 | Fox et al. | |
| 6,061,741 A | | 5/2000 | Murphy, Jr. et al. | |
| 6,078,912 A | * | 6/2000 | Buerger et al. | 705/8 |
| 6,167,408 A | * | 12/2000 | Cannon et al. | 707/203 |
| 6,185,601 B1 | * | 2/2001 | Wolff | 709/203 |
| 6,240,407 B1 | | 5/2001 | Chang et al. | |
| 6,324,544 B1 | | 11/2001 | Alam et al. | |
| 6,330,600 B1 | * | 12/2001 | Matchefts et al. | 709/223 |
| 6,334,178 B1 | * | 12/2001 | Cannon et al. | 707/201 |
| 6,345,308 B1 | * | 2/2002 | Abe | 709/248 |
| 6,363,377 B1 | | 3/2002 | Kravets et al. | |
| 6,404,445 B1 | * | 6/2002 | Galea et al. | 709/203 |
| 6,507,813 B2 | * | 1/2003 | Veditz et al. | 717/137 |

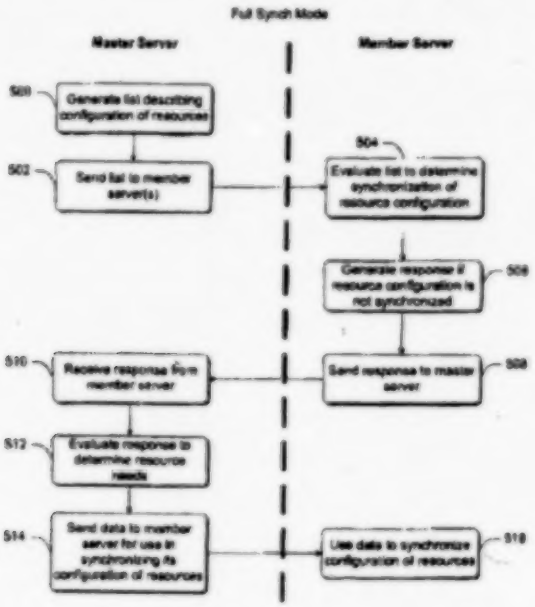
* cited by examiner

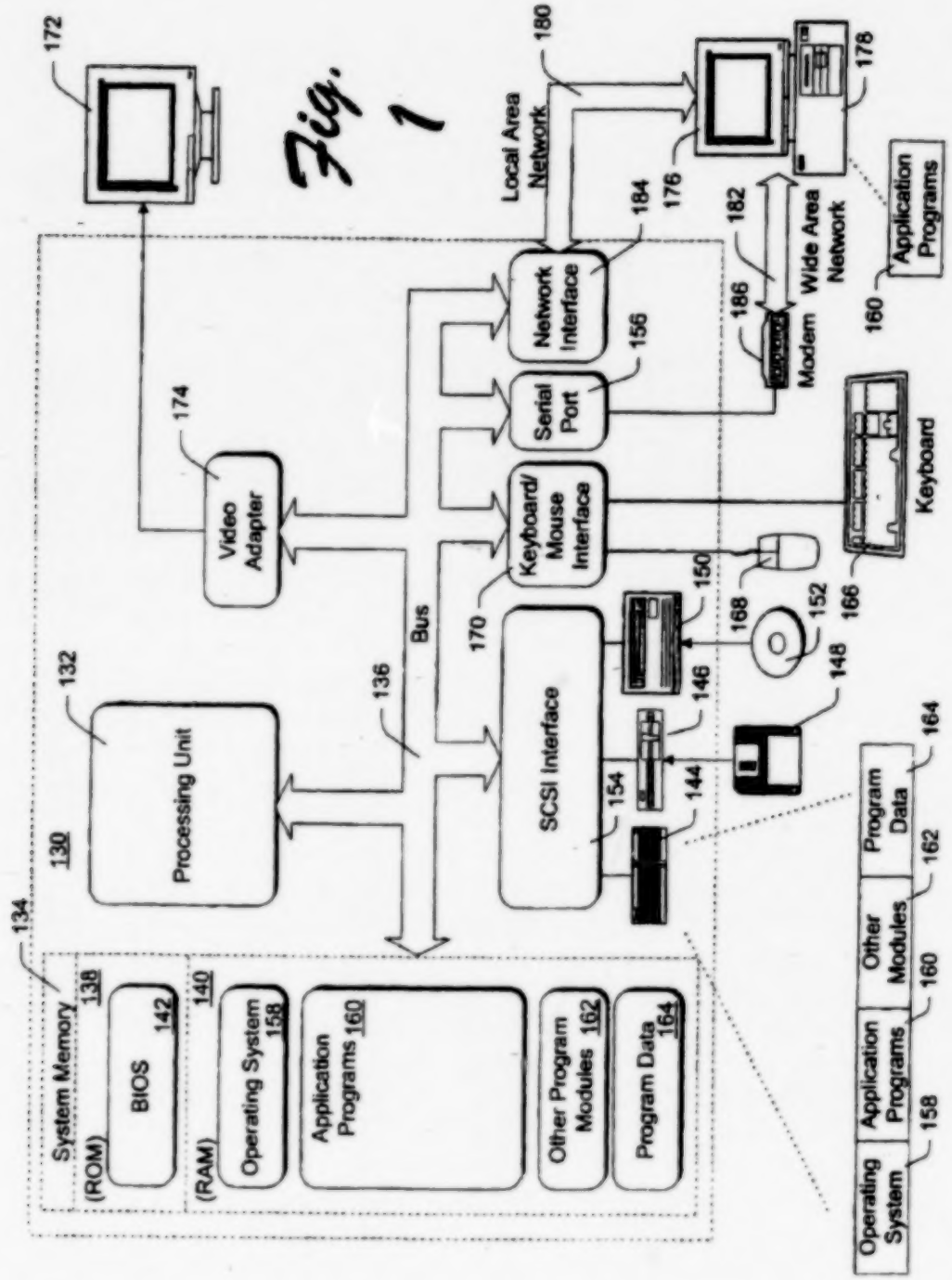
*Primary Examiner*—David Wiley
*Assistant Examiner*—Joseph E Avellino
(74) *Attorney, Agent, or Firm*—Lee & Hayes PLLC

(57) **ABSTRACT**

Methods and systems for synchronizing a configuration of resources across multiple computing devices are described. Associated with the master computing device is a configuration of resources that are desired to be synchronized across the additional computing devices. The master computing device is configured to develop a list that describes aspects of its configuration of resources. Additional computing devices are configured to receive the list that is developed by the master computing device and evaluate the list to determine whether its particular configuration of resources is synchronized with those of the master computing device. If a particular additional computing device's configuration of resources is not synchronized with those of the master computing device, the additional computing device prepares a second list that describes its resource configuration needs and sends the second list to the master computing device. The master computing device receives the second list, evaluates it, and sends data to the corresponding additional computing device for use in synchronizing its configuration of resources.

**56 Claims, 6 Drawing Sheets**




Fig. 1
172
Local Area Network
180
178
Application Programs
160
Wide Area Network
182
Modem
186
176
184
Network Interface
156
Serial Port
174
Video Adapter
Keyboard/ Mouse Interface
Keyboard
170
150
168
152
166
Bus
136
132
Processing Unit
130
SCSI Interface
146
148
154
144
134
System Memory
138
(ROM)
BIOS
142
140
(RAM)
Operating System
158
Application Programs
160
Other Program Modules
162
Program Data
164
Operating System
158
Application Programs
160
Other Modules
162
Program Data
164

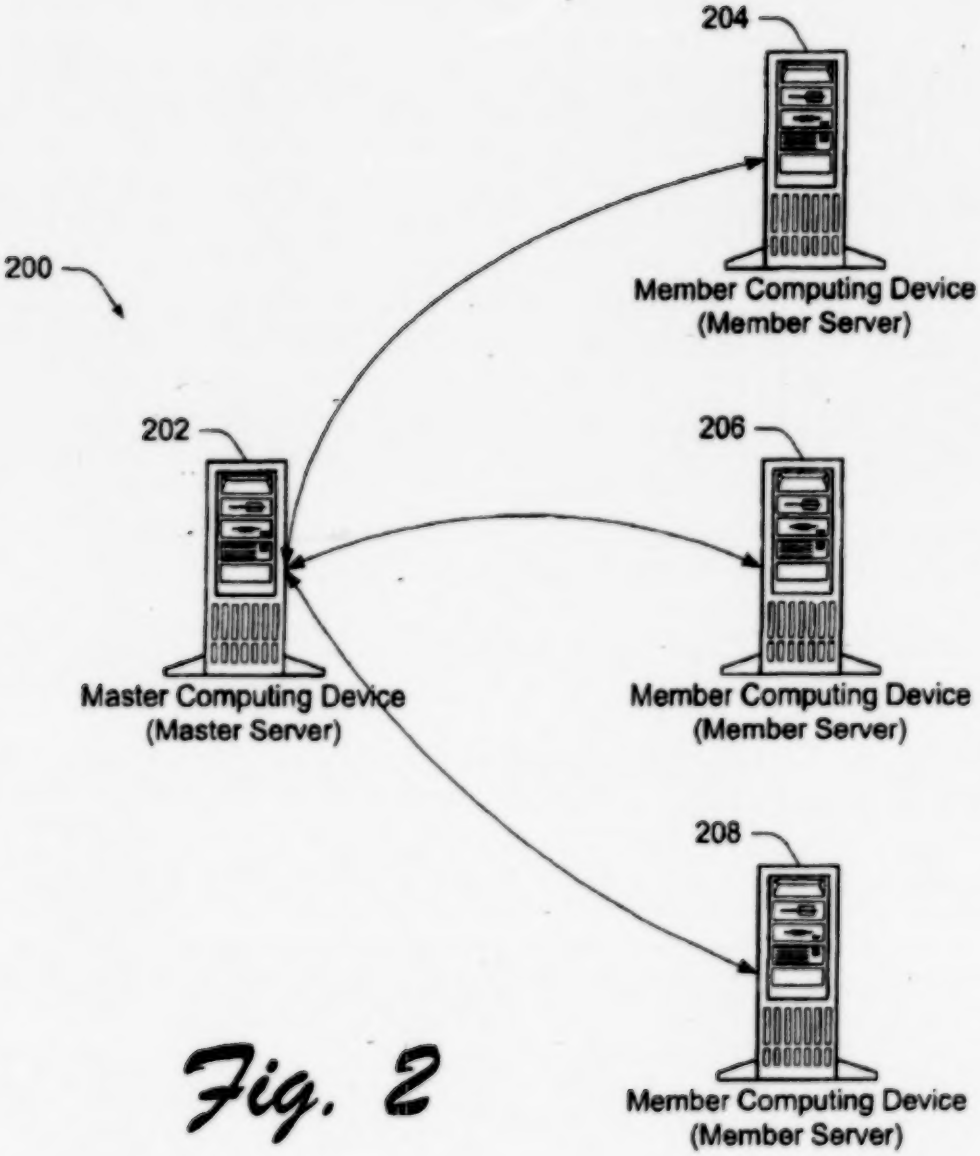

200
202
Master Computing Device
(Master Server)
204
Member Computing Device
(Member Server)
206
Member Computing Device
(Member Server)
208
Member Computing Device
(Member Server)

Fig. 2

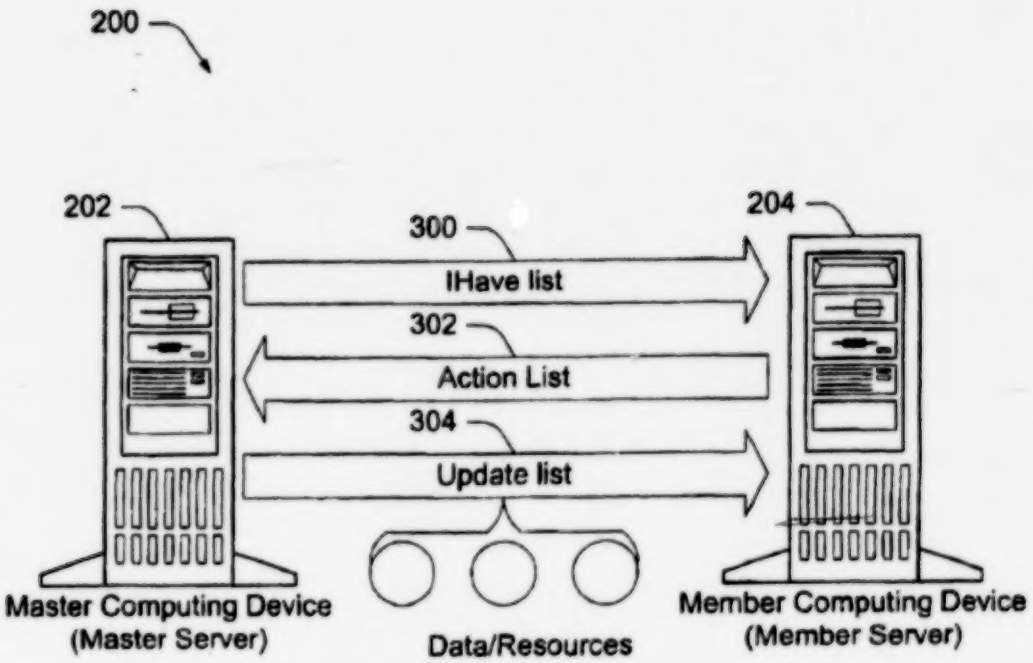

200
202
204
300
IHave list
302
Action List
304
Update list
Master Computing Device
(Master Server)
Data/Resources
Member Computing Device
(Member Server)

Fig. 3

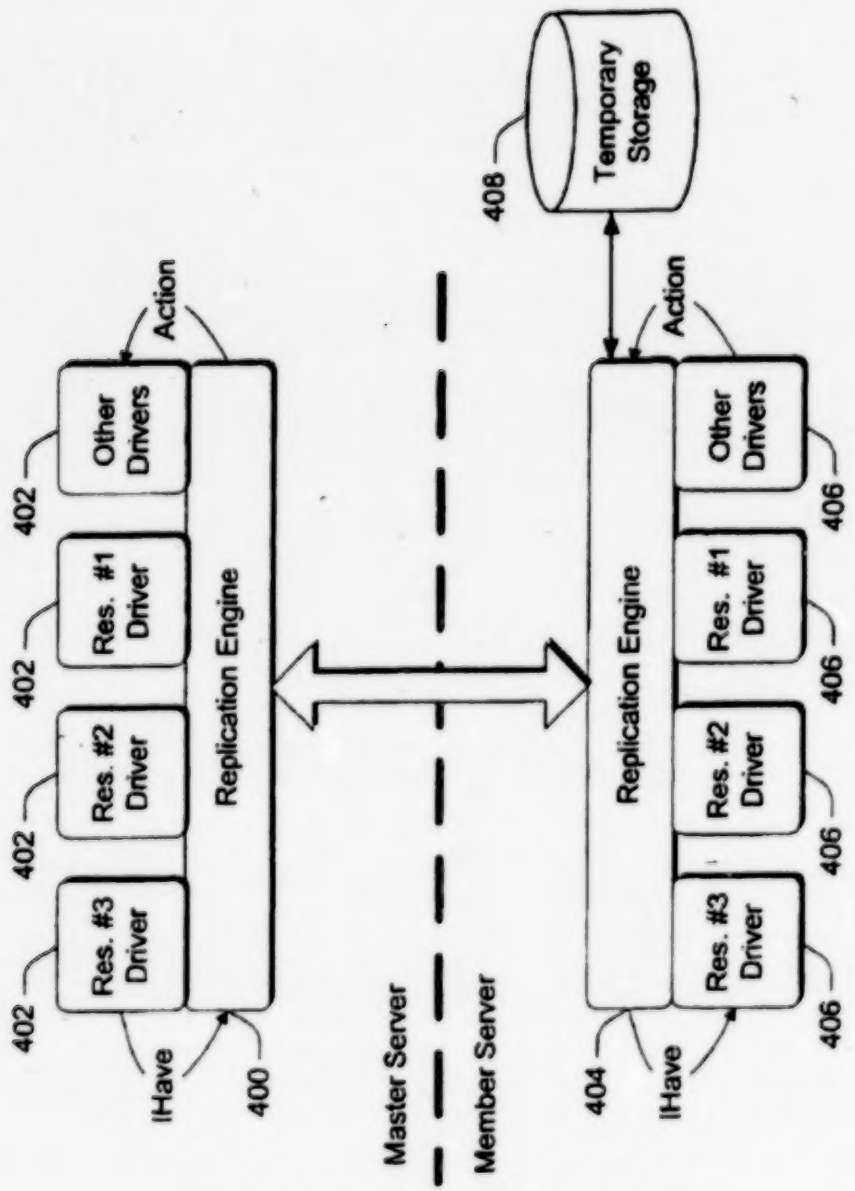

Temporary Storage
408
Action
Other Drivers
Res. #1 Driver
Res. #2 Driver
Res. #3 Driver
Replication Engine
402
IHave
400
Master Server
Member Server
404
406

Fig. 4

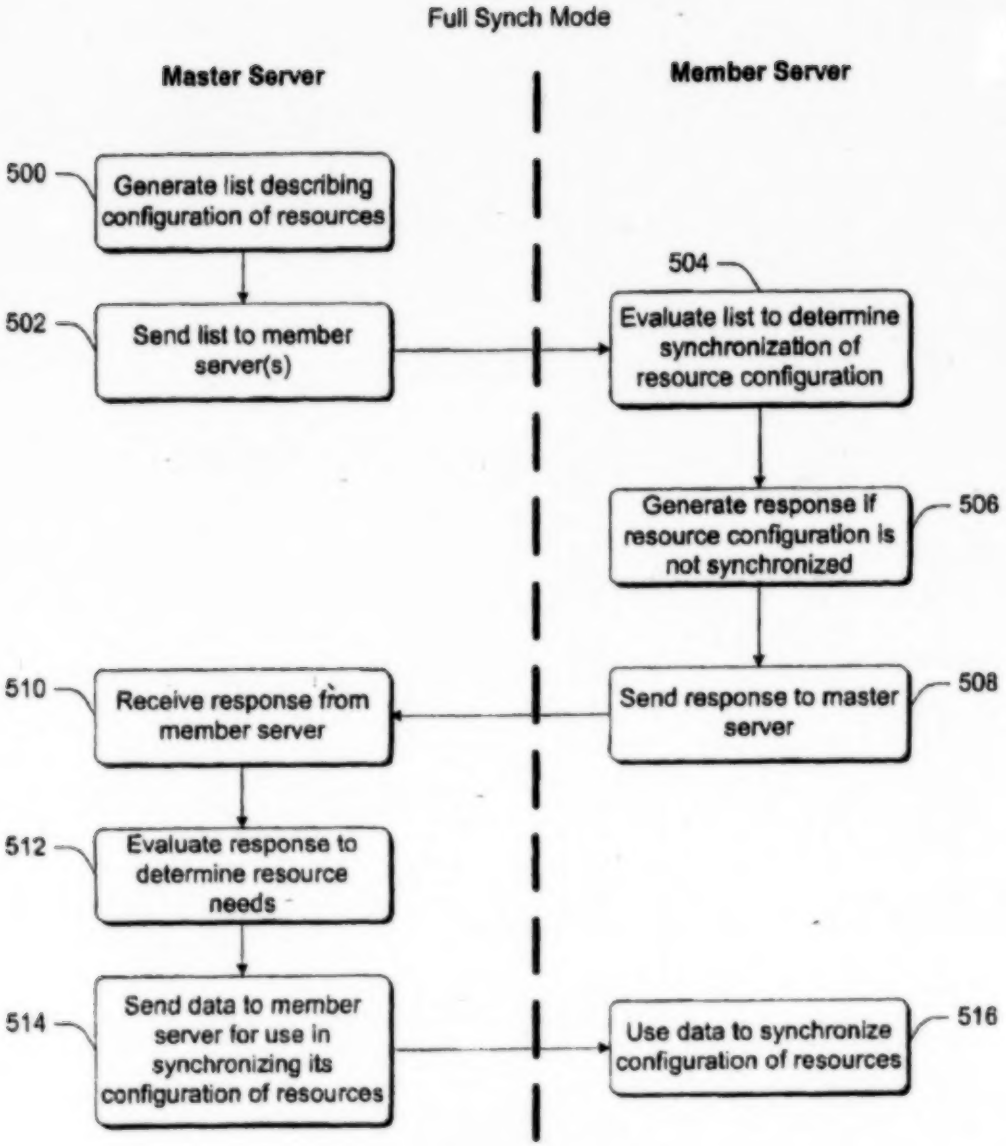

Full Synch Mode
Master Server
Member Server
500
Generate list describing configuration of resources
502
Send list to member server(s)
504
Evaluate list to determine synchronization of resource configuration
Generate response if resource configuration is not synchronized
506
Send response to master server
508
510
Receive response from member server
512
Evaluate response to determine resource needs
514
Send data to member server for use in synchronizing its configuration of resources
Use data to synchronize configuration of resources
516

Fig. 5

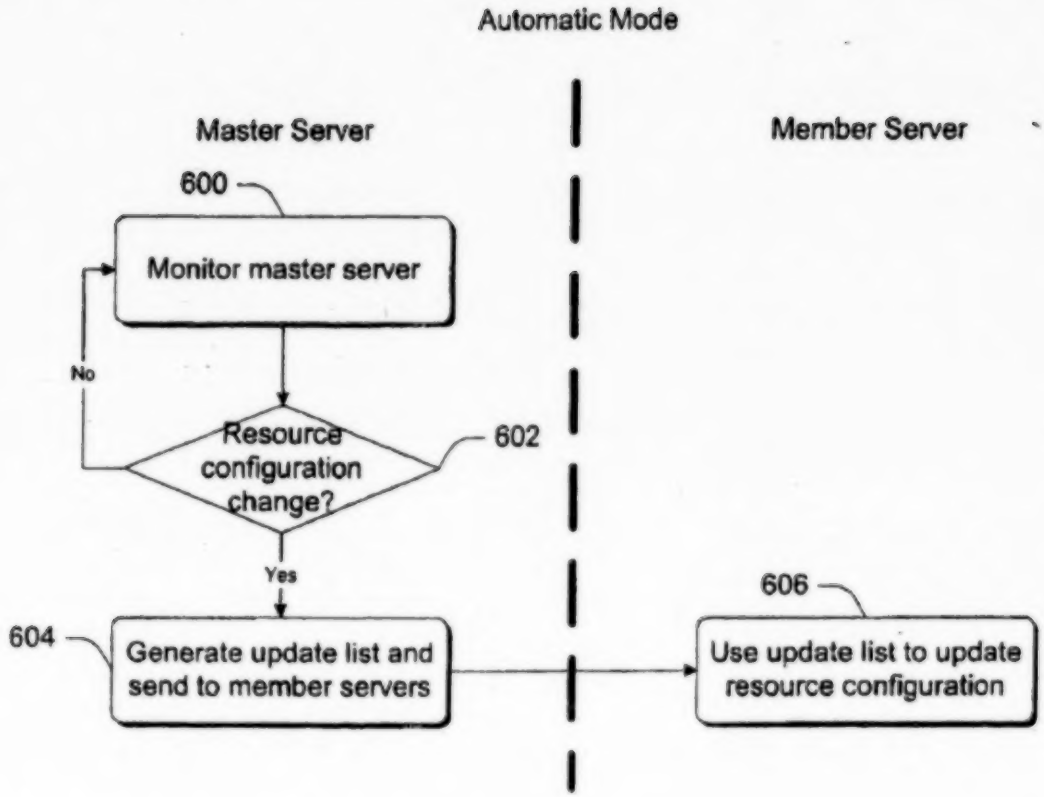

Automatic Mode
Master Server
Member Server
600
Monitor master server
No
Resource configuration change?
602
Yes
604
Generate update list and send to member servers
606
Use update list to update resource configuration

Fig. 6

# METHODS AND SYSTEMS FOR SYNCHRONIZING MULTIPLE COMPUTING DEVICES

## TECHNICAL FIELD

This invention relates to methods and systems for synchronizing multiple computing devices. In particular, the invention relates to methods and systems for synchronizing multiple different web servers.

## BACKGROUND

As the popularity of networked systems, e.g. distributed computing systems, continues to grow and find use in contemporary times, there is an ever increasing need to ensure that certain computing devices within such systems are synchronized. By "synchronized" it is meant that the configuration of resources that each computing device uses to perform its job are the same, and that any associated resource settings are standard across all of the computing devices. Specifically, many times, due to scalability issues (i.e. adding additional computing devices to service a larger or growing client base), it becomes desirable to implement a particular functionality across multiple computing devices. In these instances it is highly desirable for each different computing device to operate identically. This is important because it is likely that different clients will be serviced by different computing devices. Accordingly, it is highly desirable for each client to receive the same service and be able to accomplish the same tasks even though they are using different computing devices. In order for the computing devices to operate identically, the pertinent resource configurations should be the same across all of the computing devices. Yet, often times this is not the case. When the configuration of resources is not standardized, inconsistent results can occur.

As a more concrete example, consider the case of a computing device in the form of a server that is programmed to function as a Web server. The World Wide Web or "Web" works on a client/server model in which client software—known as a Web browser—runs on a local computer. The server software runs on a Web server. Client machines effect transactions with Web servers using Hypertext Transfer Protocol (HTTP), which is a known application protocol providing users with access to files (e.g., text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML). HTML provides basic document formatting and allows the developer to specify "links" to other servers and files. In the Internet paradigm, a network path to a server is identified by a so-called Uniform Resource Locator (URL) having a special syntax for defining a network connection. Use of an HTML-compatible browser (e.g., Internet Explorer) at a client machine involves specification of a link via the URL. In response, the client makes a request to the server identified in the link and receives in return a document formatted according to HTML. The Web server is usually a standalone file server that services various Web document requests. One particular type of Web server is Microsoft's Internet Information Server (IIS).

One of the problems that has arisen, and one which is particularly illustrated in the context of Web servers, is that customers often need to scale Web servers so that its services are widely available and highly reliable. One way of doing this is to incorporate multiple servers into a networked system. Each of the individual servers should operate identically in order for the same services to be rendered regardless of which server is used to the render services. In order for the servers to operate identically, their resource configurations should be the same. If the resource configurations are not the same, then inconsistent results can occur. Computing devices such as Web servers can typically have many different types of resources and settings. As an example, Microsoft's IIS has resources that include, but are not limited to files, metabase entries, registry keys, and cryptographic functions. Current solutions are inadequate to solve the problems of synchronizing multiple servers. For example, there are solutions that copy files from one server to another, or that replicate a resource setting from one machine to another, but these solutions fall far short synchronizing non-file resources and, in particular, multiple different resources and settings.

Accordingly, this invention arose out of concerns associated with providing systems and methods for synchronizing resource configurations across multiple computing devices.



## SUMMARY

Methods and systems for synchronizing a configuration of resources across multiple computing devices are described. In the described embodiment, a computer network comprises a master computing device and a plurality of additional computing devices communicatively linked with the master computing device. The master computing device comprises a configuration of resources that are desired to be synchronized across the additional computing devices. The master computing device is configured to develop a set of lists that describe aspects of its configuration of resources. Each additional computing device is configured to receive the lists that are developed by the master computing device and evaluate the lists to determine whether its particular configuration of resources are synchronized with those of the master computing device. If a particular additional computing device's configuration of resources is not synchronized with those of the master computing device, the additional computing device prepares a second set of lists that describe its resource configuration needs and sends the second set of lists to the master computing device. The master computing device receives the second list set, evaluates it, and sends data to the corresponding additional computing device for use in synchronizing its configuration of resources. Each of the additional computing devices can send separate independent lists that describe their resource needs and the master computing device is capable of individually responding to each one. In one aspect, the computing devices are computer servers that are configured as Web servers.

One aspect provides a server software architecture that comprises a replication engine to handle communication between a master server and a plurality of member servers. At least one driver is provided and is capable of communicating with the replication engine. The driver is associated with a particular resource type (a.k.a. store) that is accessible from the master server. Resources could potentially be stored on a remote location (e.g., a remote file system). In the described embodiment, each resource type of interest has its own separate driver that is programmed to interact with it. Each driver is configured to prepare lists that describe characteristics of its associated resource and articulate the lists to the replication engine. The replication engine is configured to receive the lists from the driver(s) and send the lists to one or more of the member servers so that the member servers can use the lists to synchronize a resource that corresponds to the resource with which the driver(s) is

associated. In the described embodiment, each of the member servers has an architecture that mirrors the architecture of the master server. Accordingly, each member server has a replication engine to handle communication with the master server, and a collection of member servers drivers that are individually associated with different resource types that are resident on the member server. Each of the member server drivers is configured to evaluate the resources with which it is associated in response to its replication engine receiving a list from the master server's replication engine. The member server driver then determines whether its resources are synchronized with a corresponding resource on the master server. The member server driver is further configured to prepare a response to the master server that describes resource needs of the member server in the event that a resource is determined not to be synchronized with the corresponding resource on the master server. The master server receives the response with its replication engine and passes the response to the corresponding master server driver that is associated with the resource of interest. The master server driver then prepares a reply to the response that includes data that can be used by the member server driver to update or synchronize its resource type. The reply is sent to the member server and processed by the member server's replication engine. The replication engine passes the reply to the appropriate driver so that the driver can synchronize the resource of interest.

Another aspect automatically synchronizes resource configurations across multiple computing devices so that any time a resource configuration is updated or changed on, a master computing device, those changes get promulgated to a plurality of member servers. In the described embodiment, promulgation of the resource configuration changes takes place in a list-based manner.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a diagram of an exemplary computing device that can be utilized to implement one or more computing devices in accordance with the described embodiment.

FIG. **2** is a network diagram that illustrates exemplary computing devices in the form of a master server and a plurality of member servers in accordance with the described embodiment.

FIG. **3** is a diagram that illustrates an exemplary information exchange between a master computing device and a member computing device in accordance with the described embodiment.

FIG. **4** is a diagram of an exemplary software architecture in accordance with the described embodiment.

FIG. **5** is a flow diagram that describes steps in a method in accordance with the described embodiment.

FIG. **6** is a flow diagram that describes steps in a method in accordance with the described embodiment.

## DETAILED DESCRIPTION

### Exemplary Computing Device

Preliminarily, FIG. **1** shows a general example of a computer **130** that can be used in accordance with the described embodiment. Computer **130** can be utilized to implement various computing devices in accordance with the described embodiment.

Computer **130** includes one or more processors or processing units **132**, a system memory **134**, and a bus **136** that couples various system components including the system memory **134** to processors **132**. The bus **136** represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. The system memory **134** includes read only memory (ROM) **138** and random access memory (RAM) **140**. A basic input/output system (BIOS) **142**, containing the basic routines that help to transfer information between elements within computer **130**, such as during start-up, is stored in ROM **138**.

Computer **130** further includes a hard disk drive **144** for reading from and writing to a hard disk (not shown), a magnetic disk drive **146** for reading from and writing to a removable magnetic disk **148**, and an optical disk drive **150** for reading from or writing to a removable optical disk **152** such as a CD ROM or other optical media. The hard disk drive **144**, magnetic disk drive **146**, and optical disk drive **150** are connected to the bus **136** by an SCSI interface **154** or some other appropriate interface. The drives and their associated computer-readable media provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for computer **130**. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk **148** and a removable optical disk **152**, it should be appreciated by those skilled in the art that other types of computer-readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk **144**, magnetic disk **148**, optical disk **152**, ROM **138**, or RAM **140**, including an operating system **158**, one or more application programs **160**, other program modules **162**, and program data **164**. A user may enter commands and information into computer **130** through input devices such as a keyboard **166** and a pointing device **168**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are connected to the processing unit **132** through an interface **170** that is coupled to the bus **136**. A monitor **172** or other type of display device is also connected to the bus **136** via an interface, such as a video adapter **174**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

Computer **130** commonly operates in a networked environment using logical connections to one or more remote computers, such as a remote computer **176**. The remote computer **176** may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer **130**, although only a memory storage device **178** has been illustrated in FIG. **1**. The logical connections depicted in FIG. **1** include a local area network (LAN) **180** and a wide area network (WAN) **182**. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, computer **130** is connected to the local network **180** through a network interface or adapter **184**. When used in a WAN networking environment, computer **130** typically includes a modem **186** or other means for establishing communications over the wide area network **182**, such as the Internet. The modem **186**, which may be internal or external, is connected to the

bus **136** via a serial port interface **156**. In a networked environment, program modules depicted relative to the personal computer **130**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Generally, the data processors of computer **130** are programmed by means of instructions stored at different times in the various computer-readable storage media of the computer. Programs and operating systems are typically distributed, for example, on floppy disks or CD-ROMs. From there, they are installed or loaded into the secondary memory of a computer. At execution, they are loaded at least partially into the computer's primary electronic memory. The invention described herein includes these and other various types of computer-readable storage media when such media contain instructions or programs for implementing the steps described below in conjunction with a microprocessor or other data processor. The invention also includes the computer itself when programmed according to the methods and techniques described below.

For purposes of illustration, programs and other executable program components such as the operating system are illustrated herein as discrete blocks, although it is recognized that such programs and components reside at various times in different storage components of the computer, and are executed by the data processor(s) of the computer.

## Overview

The methods and systems described below provide a simple, flexible approach to maintaining a configuration of resources synchronized across multiple computing devices. For purposes of this document, a "configuration of resources" will be understood to include either or both of a collection of resources that a computing device utilizes to implement its functionality, or various resource settings that can be set relative to a resource. Although the described embodiment is described in the context of a computing device that comprises a server and, in particular a web server, it is to be understood that the principles of the described invention should not be so limited. Accordingly, the principles described below are applicable to various other types of computing devices.

For purpose of understanding what constitutes an exemplary "resource", consider Microsoft's Internet Information Server (IIS). There, resources can include, without limitation, files, metabase entries, registry keys, and cryptographic functionality (e.g. certificate stores and the like). A "setting" can constitute any type of setting for a resource. For example, consider that a user creates an application and desires to store the number of threads to use for that application. The "number of threads" can be considered as a "setting" that might typically be stored in the computer's registry (a store).

One advantage of the described embodiment is its ability to promulgate resource configuration changes for any type of resource throughout system of multiple computing devices. The inventive systems and methods are extensible and provide a degree of flexibility that far exceeds the capabilities of other attempts at synchronization (i.e. merely copying files from one machine to another).

In the described embodiment, two different modes can be utilized to ensure that resources across multiple computing devices remain synchronized. A so-called "full synch" mode can be used to initialize a plurality of different machines to synchronize to a master computing device. This ensures that all of the computing devices in a particular network configuration are standardized. After that an "automatic" mode monitors the master computing device so that any time a change is made to one of its resources, that change is promulgated down through the other computing devices. This, as the name implies, happens automatically without user intervention. In addition, the "full-synch" mode can be utilized to periodically update or "resync" the computing devices so that any resources that have been changed (other than resources on the master computing device) can be placed back into a state which is synchronized with that of the master computing device.

FIG. **2** shows a computer network **200** which, in accordance with the described embodiment comprises a master computing device **202** and a plurality of additional computing devices **204, 206, 208** communicatively linked with master computing device **202**. The master computing device **202** comprises a configuration of resources that are desired to be synchronized across the additional computing devices **204–208**. In the "full synch" mode, the master computing device **202** develops lists that describes aspects of its configuration of resources and sends the lists to the additional computing devices **204–208**. Each additional computing device **204–208** receives the lists and evaluates the lists to determine whether its particular configuration of resources are synchronized with those of the master computing device. If a particular additional computing device's configuration; of resources is not synchronized with those of master computing device **202**, the additional computing device prepares a second set of lists that describes its resource configuration needs and sends, the second set of lists to master computing device **202**. The communication between the master computing device and the additional or member computing devices is indicated by the double-headed arrows in FIG. **2**. The master computing device **202** receives the second set of lists, evaluates it, and sends data (e.g. in the form of resources or settings) to the corresponding additional computing device for use in synchronizing its configuration of resources. Each of the additional computing devices **204–208** can send separate independent lists that describe their own particular resource needs and the master computing device **202** is capable of individually responding to each one. In the "automatic" mode, changes in the master computing device's configuration of resources are detected and then data is sent to each of the additional computing devices **204–208** for use in synchronizing its configuration of resources.

One particularly advantageous aspect of the described embodiment is that communication takes place between the master computing device and the additional computing devices through the use of extensible markup language (XML) techniques. Here, XML lists can be sent back and forth between the master computing device and the additional computing devices. The XML lists can either contain embedded update material for use in synchronizing resource configurations, or can reference particular resources that are to be updated using data that is either previously or subsequently sent. It should be appreciated that any format that is capable of describing a hierarchical list of objects would do and that the use of XML represents but one exemplary way of doing so.

One aspect provides a server software architecture that facilitates communication between a master server and additional member servers. In the described embodiment, a replication engine is provided and handles communication between the master server and the member servers. At least

one driver is provided and is capable of communicating with the replication engine. The driver is associated with a particular resource type that is accessible from the master server. In the described embodiment, each resource type of interest (there can be many resource types) has its own separate driver that is programmed to interact with it. Each driver is responsible for reading and writing its own content type, as well as other functions. For example, each driver is responsible for preparing a set of lists that describes characteristics of its associated resource and articulating the lists to the replication engine. The replication engine receives the lists from the driver(s) and sends the lists to one or more of the member servers **204–208** so that the member servers can use the lists to synchronize a resource that corresponds to the resource with which the driver(s) is associated.

In the described embodiment, each of the member servers comprises a software architecture that mirrors the software architecture of the master server. Accordingly, each member server has a replication engine to handle communication with the master server, and a collection of member server drivers that are individually associated with different resources that are resident on the member server. Each of the member server drivers evaluates the resource with which it is associated in response to its replication engine receiving a list from the master server's replication engine. The member server driver determines whether its associated resource is synchronized with a corresponding resource on the master server. The member server driver prepares a response to the master server that describes its resource needs in the event that a resource is determined not to be synchronized with the corresponding resource on the master server. In the described embodiment, each member server responds separately, with the separate responses being queued for transmission as one unit for greater efficiency. The master server receives the response with its replication engine and passes the response to the corresponding master server drivers that are associated with the resources of interest. The master server driver prepares a reply to the response that includes data that can be used by the member server driver to update or synchronize its resource. The reply is sent to the member server and processed by the member server's replication engine. The replication engine passes the reply to the appropriate member server driver so that the driver can synchronize the resource of interest.

### List-Based Exchange

FIG. **3** shows master computing device or master server **202** and member computing device or member server **204**, and an exemplary communication exchange between them in more detail. The communication exchange facilitates synchronizing the member servers. In the described embodiment, member server resource configurations are synchronized with resource configurations of the master server through the use of one or more lists that are passed between the servers. Any suitable communication or list can be used in the information exchange between the servers, with an exemplary type of list being discussed below in the section entitled "Extensible Markup Language Lists".

In the illustrated example, master server **202** comprises a configuration of resources that define the content and settings for the master server. Exemplary content and settings were mentioned above. The master server's configuration of resources is desired to be promulgated to other servers, e.g. member server **204**, so that the master and member servers can implement a common functionality. In this example, the common functionality that is implemented by the master and member servers is that of a web server. Other functionalities can, of course, be implemented.

The master server **202** is configured to develop a list that describes aspects or characteristics of its configuration of resources. In the illustrated example, an exemplary list **300** is shown as an "IHave" list. This list describes the resources that the master server has—hence the name "IHave". Exemplary aspects or characteristics can include, without limitation, individual signatures that represent one or more of the particular resources. For example, a hash can be made of a particular resource, e.g. a file, and thus constitutes the signature for that resource. Another example of a signature is a date/time stamp. Any type of suitable aspect or characteristic can be used and incorporated in the list that is developed by the master server. In many instances it is desirable to use an aspect or characteristic that represents the resource of interest, yet takes up less room, e.g. in a buffer, than the resource. This enables faster processing than if the entire resource were used. It is possible in some circumstances, for the aspect or characteristic to comprise the resource itself.

Each of the member servers **204** is communicatively linked with the master server **202** and comprises a configuration of resources that correspond, at least in part, to the configuration of resources of the master server **202**. For example, it is possible in some instances for the configuration of resources on one or more of the member servers to have been changed for whatever reason. In this instance, the member server's configuration of resources would not correspond entirely with the master server's configuration of resources. As another example, it is possible for the master server's configuration of resources to have been changed, e.g. adding or deleting a resource. In each of these instances it is highly desirable for the master server's configuration of resources to be promulgated to the member servers so that standardization is preserved.

Each of the member servers **204** is configured to receive the list that is developed by the master server **202** and evaluate the list to determine whether its particular configuration of resources are synchronized with those of the master server. For example, each member server can use the aspects or characteristics that are contained in the list to evaluate their own resources. For example, where the list contains a hash of one or more resources, a member server can make a hash of its corresponding resources using the same hashing algorithm and then simply compare the hashes. Alternately, if a date/time stamp is used, the member server can simply compare the date/time stamps of its associated resources with the date/time stamps contained in the list. If any differences are found during this evaluation, the member server will know that one or more of its resources are not synchronized with the corresponding resource on the master server **202**.

In the event that one or more of the member servers **204** determines, through its evaluation of the list provided by the master server **202**, that its configuration of resources is not synchronized, the member server is configured to communicate any of its resource needs to the master server **202**. In the illustrated example, the member server **204** does so by building a second list **302**, referred to in the illustration as an "Action list". The Action list **302** describes the resource or resource configuration needs of the member server. The member server **204** sends the Action list **302** to the master server **202** for evaluation. The Action list **302** requests data in the form of resources or resource settings that will enable the member server to synchronize itself with the configuration of resources of the master server **202**. When the master server **202** receives this communication from the member servers, it evaluates the Action list **302** to determine the

resource configuration needs of the particular individual member servers. Responsive to its evaluation of the Action list **302**, the master server **202** sends the necessary needed data to the member server **204** so it can synchronize its configuration of resources with that of the master server. In the illustrated example, the data that is sent from the master server to the member server can be sent in an Update list **304**. The Update list can contain data that describes various resource configuration settings (e.g. the number of threads that a particular application can use) as well as data that represents the resources themselves (e.g. files, certificates and the like). It is also possible for the Update list to merely describe the resources that are going to be sent, with the resources being separately sent in a different communication between the master server **202** and the member server **204** (e.g. large files that might be more economically sent using a different protocol).

In the described embodiment, member servers can be updated in a couple of different ways. In a "full synch" mode, the master server initially sends an IHave list **300** to each of the member servers for the server to do their own evaluation. The member servers evaluate their resource configurations in view of the list and take any measures that are necessary to ensure that they are synchronized with the master server. This process was described immediately above. The full synch mode can advantageously be used to initialize the member servers. Another mode referred to as an "automatic mode" is used by the master server to automatically update the member servers with resource specific changes. For example, if a particular resource is modified on the master server, data associated only with the changed resource can be sent to the member servers so that they can individually update their corresponding resource.

## Extensible Markup Language Lists

As mentioned above, communication between the master server and the member server can take place in any suitable manner. It has been found particularly advantageous, however, to use lists that are defined in terms of extensible markup language (XML). The use of XML greatly facilitates generating, handling, and passing the lists back and forth between the master and member servers.

Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML is similar to HTML in that it is a tag-based language. By virtue of its tag-based nature, XML defines a strict tree structure or hierarchy. XML is a derivative of Standard Generalized Markup Language (SGML) that provides a uniform method for describing and exchanging structured data in an open, text-based format. XML utilizes the concepts of elements and namespaces. Compared to HTML, which is a display-oriented markup language, XML is a general purpose language for representing structured data without including information that describes how to format the data for display.

XML "elements" are structural constructs that consist of a start tag, an end or close tag, and the information or content that is contained between the tags. A "start tag" is formatted as "<tagname>" and an "end tag" is formatted as "</tagname>". In an XML document, start and end tags can be nested within other start and end tags. All elements that occur within a particular element must have their start and end tags occur before the end tag of that particular element. This defines a strict tree-like structure. Each element forms a node in this tree, and potentially has "child" or "branch" nodes. The child nodes represent any XML elements that occur between the start and end tags of the "parent" node.

XML accommodates an infinite number of database schemas. Within each schema, a "dictionary" of element names is defined. The dictionary of element names defined by a schema is referred to as a "namespace." Within an XML document, element names are qualified by namespace identifiers. When qualified by a namespace identifier, a tag name appears in the form "[namespace]:[tagname]". This model enables the same element name to appear in multiple schemas, or namespaces, and for instances of these duplicate element names to appear in the same XML document without colliding. Start tags can declare an arbitrary number of "attributes" which declare "property values" associated with the element being declared. Attributes are declared within the start tag using the form "<[tagname] [attribute1] [attribute2]. . . [attributeN]>", where an attribute1 through attributeN are declarations of an arbitrary number of tag attributes. Each attribute declaration is of the form "[attributeName]=["attributeValue"]" where each attribute is identified by a unique name followed by an "=" character, followed by the quoted value of the attribute.

## Exemplary IHave List

Recall that the IHave list is used by the master server to describe the resources or configuration of resources that each of the member servers should have. The IHave list is built by the master server and then sent to each of the member servers so that they can evaluate their own content and settings. In the described embodiment, the IHave list can include the following selected tags and attributes. It is to be understood that the following table constitutes but one way of implementing the IHave list using XML. Accordingly, other ways of implementing the IHave list either using or not using XML are possible. Optional tags or attributes are enclosed in "[ ]".

| Tag | Attribute | Explanation |
|---|---|---|
| <SCOPE> | RESTYPE | Resource type |
| | [ENUM] | Either "partial" or "full". This indicates whether the interaction of all items and subnodes is complete. If ENUM is "full", an item or node that is not in this list should be deleted from the member server. If ENUM is "partial" then an item or node that is not in the list is not to be deleted. |
| | ROOT | Defines the absolute root path. |
| | TYPE | Defines the type of list, e.g. IHAVE, ACTION, UPDATE. |
| | [DIFFUSION] | Either "unicast" or "multicast". Indicates how the list is to be dispensed. "Unicast" lists get sent to only one member server while "multicast" lists get sent to many member servers. |
| | [LAST] | Either "0" or "1". Indicates whether this is the last of a list sequence or not. |
| | [ETAGTYPE] | This is utilized to ensure that the content is the same with a high degree of probability. This controls the tradeoff between ETAG computation cost and the probability of generating the same ETAG for different content. Values can include "strong" and "weak" |
| | ITEMSIZE | Total size of all data associated with all items. |
| | ITEMCOUNT | Count of all items. |

-continued

| Tag | Attribute | Explanation |
|---|---|---|
| <NODE> | | This tag essentially defines a hierarchy under the root or parent. |
| | SUBPATH | Sub path relative to the father node. |
| | [ENUM] | Either "partial" or "full". If "full", then only data defined by the sub-tree exists, i.e. everything else is to be deleted. |
| <ITEM> | | Defines the granularity of the replication, e.g. a file. |
| | NAME | Name inside the node. |
| | ETAG | Opaque signature of a resource. Can be based on a hash, date/time, etc. |
| | SIZE | Size of all the data associated with this item. |

As an exemplary portion of an IHave list that incorporates the above tags and some or all of the attributes, consider the following:

```
<SCOPE TYPE="IHAVE" LAST="0" RESTYPE="FS" ROOT="c:\inetpub"
       ETAGTYPE="STRONG"   ITEMSIZE="8810"   ENUM="FULL"
       ITEMCOUNT="2" DIFFUSION="MULTICAST"
   <NODE SUBPATH="wwwroot" ENUM="FULL">
    <ITEM NAME="default.asp" ETAG="67ED89B" SIZE="345"></ITEM>
    <ITEM NAME="myimg.jpg" ETAG="5656365" SIZE="8465"></ITEM>
    <NODE SUBPATH="Images"/>
    <NODE SUBPATH="Apps"/>
   </NODE>
</SCOPE>
<SCOPE TYPE="IHAVE" RESTYPE="FS" ROOT="c:\inetpub\wwwroot"
       ETAGTYPE="STRONG"   ITEMSIZE="4"   ITEMCOUNT="1"
       DIFFUSION="MULTICAST"
   <NODE SUBPATH="Images" ENUM="FULL">
    <ITEM NAME="Images.jpg" ETAG="46ED89B" SIZE="4"></ITEM>
   </NODE>
</SCOPE>
```

The exemplary IHave list describes file resources that the master server has in the root "c:/inetpub". Here, the XML list is the last of the XML lists that are being sent. The resource type ("FS") with which the list is associated is a file resource. The RESTYPE is used to identify how and where the list is processed. This will become more apparent below as one exemplary software architecture is described. There are two items (i.e. files) in the subroot "c:/inetpubwwwroot" that are of interest. In addition, there are two subpaths or subdirectories (i.e. Images and Apps). The enumeration for the root "c:/inetpub" is "full" which means that if a particular resource is not described in the IHave list, it is to be deleted from the member server. With respect to the individual file resources, a first of the files is "default.asp" and a second of the files is "myimg.jpg". Each of the files has an associated ETAG which is a signature that represents the file. The signature is compared by the member server to its own corresponding file resource signature to determine whether its file resource is synchronized with the master server resource. The subpath "Images" (represented by the second portion of the XML list) includes one file "Images.jpg" having the indicated ETAG and size.

## Exemplary Action List

Once the member server has received the IHave list from the master server and evaluated its resources accordingly, it prepares an Action list that describes its resource needs for the master server. That is, if any of the subject resources are determined to be unsynchronized with the master server's resources, then the member server uses the Action list to articulate its resource needs to the master server. The, selected tags and attributes for the Action list are generally the same as the tags and attributes for the IHave list. The Action list does not, however, contain an ETAG attribute. As an exemplary Action list that responds to the IHave list above, consider the following:

```
<SCOPE TYPE="ACTION" LAST="0" RESTYPE="FS" ROOT="c:\inetpub"
       ITEMSIZE="8810" ENUM="FULL" ITEMCOUNT="2"
   <NODE SUBPATH="wwwroot" ENUM="FULL">
    <ITEM      NAME="default.asp"          SIZE="345"
UPDATETIME="7435:73853"><STREAM/></ITEM>
   </NODE>
</SCOPE>
<SCOPE TYPE="ACTION" RESTYPE="FS" ROOT="c:\inetpub\wwwroot"
       ITEMSIZE="0" ITEMCOUNT="1">
   <NODE SUBPATH="Images" ENUM="FULL">
    <ITEM NAME="Images2.jpg"><ATTRIBUTES/></ITEM>
   </NODE>
</SCOPE>
```

In the above example, a member server has determined based upon its evaluation of the IHave list that it needs two

things to be synchronized. First it needs the file "default.asp", and second it needs the file-system attribute associated with the file "Images2.jpg". The "stream" designation indicates that the member server has requested the actual file itself. Once the member server has built the Action list, it sends the Action list to the master server so that the master server can evaluate the resource needs to the member server and respond to them accordingly.

### Exemplary Update List

After the master server receives the Action lists from a member server, it evaluates the member server's resource needs and responds to the member server. In the illustrated example, the master server responds with an Update list. With the Update list, the master server can either embed the needed resource data in the list itself, or, simply reference resource data that was or is sent through an alternate means. As an example, consider the exemplary Update list below that responds to the above Action list.

```
<SCOPE TYPE="UPDATE" LAST="0" RESTYPE="FS" ROOT="c:\inetpub"
        ITEMSIZE="8810" ENUM="FULL" ITEMCOUNT="2"
    <NODE SUBPATH="\wwwroot" ENUM="FULL">
     <ITEM      NAME="default.asp"              SIZE="345"
UPDATETIME="7435:73853"><STREAM LINK="\wwwroot\default.asp"/>
    </ITEM>
    </NODE>
</SCOPE>
<SCOPE TYPE="UPDATE" RESTYPE="FS" ROOT="c:\inetpub\wwwroot"
        ITEMSIZE="0" ITEMCOUNT="1">
    <NODE SUBPATH="\Images" ENUM="FULL">
     <ITEM
NAME="Images2.jpg"><ATTRIBUTES>6574</ATTRIBUTES></ITEM>
    </NODE>
</SCOPE>
```

Here, the Update list responds to the Action list with the requested file "default.asp" and the requested attribute "6574". The requested file has been sent, in this example, through an alternate file transport mode. Only a link to the location where the alternate file transport will put the file content is stored. Thus, in this example, the Update list does not have any of the file data embedded within the XML tags. The attribute data, however, is embedded directly in the Update list between the "attributes" tags.

At this point in the process, all of the data that the member server needs has been prepared by the master server and sent to the member server. The member server can now take steps to ensure that the data is properly applied for synchronization.

### Exemplary Architecture

When addressing the problem of synchronizing resource configurations among multiple different member servers, consideration must be given to the issue of work factorization. That is, how should the work that needs to be done to synchronize the servers actually get done? A balance or tradeoff can be achieved between the need to have resource-specific knowledge and the need to have general knowledge of the system.

FIG. **4** shows but one exemplary software architecture that can achieve the goal of synchronizing multiple member servers. In the illustrated example, the master server comprises a replication engine **400** and one or more master server drivers **402**. The replication engine **400** handles communication between the master server and a plurality of member servers. The replication engine **400** is also responsible for coordinating replications among the drivers **402**, authentication, error recovery, and transport. Each of the one or more master server drivers **402** is capable of communicating with the replication engine **400**. In the illustrated example, each individual driver **402** is associated with a different resource type that is accessible from the master server. The driver is responsible for actually reading and writing its resource type and comparing items within its namespace to determine whether they need to be updated. In addition, each driver is also responsible for informing the replication engine **400** when its corresponding resource has been updated and also in comparing a resource between the master server and a member server. In addition, other driver responsibilities may include such things as encryption, signature generation (and comparison), locking and unlocking the resource, failure (i.e. problems writing to the store) detection, recovery and reporting, and security.

On the member server side, the software architecture may be essentially a mirror image of the master server architecture. Specifically, the member server comprises a replication engine **404** and one or more member server drivers **406**. The collection of member server drivers **406** corresponds to the master server drivers. Accordingly, each member server driver is associated with both (a) a member server resource that corresponds to a particular resource that is resident on the master server, and (b) a master server driver that is associated with that particular resource on the master server. The replication engine **404** handles communication between the member server (i.e. member server drivers **406**) and the master server (i.e. replication engine **400**).

In the illustrated example, each of the master server drivers **402** is programmed to prepare or generate a list that describes aspects or characteristics of its associated resource type. Specifically, each master server driver **402** can prepare XML lists that are then articulated by the master server driver to the master server's replication engine **400**. There are typically multiple lists, one per namespace root per driver, e.g. FS:c:/sub1, FS:c:/sub2, etc. There are typically hundreds of lists for the file system (FS) driver. In the illustrated example, the combination of all these lists constitute the IHave list. The replication engine **400** receives a list from each driver **402**, and sends a concatenation of all lists to each of the member servers for use in synchronizing their resources.

Each member server's replication engine **404** receives the IHave lists, parses them, and sends each list to the appropriate driver **406** for processing. There may be more than one IHave list that occurs, but for the sake of simplicity, only one is described. Each of the member server drivers uses the

received list to evaluate its resource to determine whether it is synchronized with the corresponding resource on the master server. This can, in some instances, involve comparing the resource signatures with one another to determine whether there are any differences. If the evaluation determines that the resource with which the driver is associated is not synchronized with the corresponding master server resource, then the driver **406** prepares a response (i.e. an Action list) to the master server that describes its resource needs. Each driver's response portion is articulated to the replication engine **404** which formulates a combined response to the master server and then communicates the Action lists to the master server's replication engine. When the master server's replication engine receives the Action lists, it parses them and sends each of the Action list to the corresponding master server driver **402** for evaluation and processing. Each of the drivers **402** is then responsible for supplying the data that is necessary for the member server to synchronize itself. This data is sent by each driver **402** in the form of an XML list to the replication engine **400**. This data may only be a link describing a stream that is sent via a particular protocol. The replication engine **400** collects the various XML lists in the form of Update lists that are then sent to the member server's replication engine **404**. The replication engine **404** parses the Update lists and sends each list to the appropriate member server driver **406**. Each member server driver then commits the updates to a temporary storage **408** until a call is received from the master server informing the member servers to commit the changes.

Exemplary Application Program Interfaces

In the described embodiment, one or more application program interfaces (APIs) are utilized that enable the processing that is described above to take place. An application program interface comprises one or more semantically-related methods that perform particular functions. In the illustrated example, there are three groups of APIs of interest: an engine-to-engine API that enables communication between the replication engines, an engine-to-driver API that enables engine-to-driver communication, and a control API that controls the replication process. The latter API creates replication sessions, stores configurations for the replication processes and essentially controls the replication context.

IReplicationDriver

This interface is found on and implemented by each of the individual resource drivers. Each of the resource drivers is capable of generating IHave lists (master server content description), Action lists (member server requests for content) and Update lists (content updates to be applied on the member servers). This interface includes the following interface methods:

| Method | Description |
|---|---|
| DefinePath( ) | Invoked on the master server. Defines a point in a namespace on which to operate. This method is called to cause the drivers to generate an IHave list, an Action list, or an Update list depending on specified parameters. |
| ProcessIHaveList( ) | Invoked on the member servers. This method enables the drivers to process the IHave list that has been provided by the master server. The drivers return an Action list if necessary. |
| ProcessActionList( ) | Invoked on the master server. This method enables each driver to ascertain the resource needs of its corresponding member server driver. The driver returns an Update list to the engine after optionally transferring content (e.g. files) to the member server. |
| ProcessUpdateList( ) | Invoked on the member server. Adds update list to current session for commitment at a later time. |
| BeginSession( ) | Begins a new session. Multiple sessions can exist at a given time. |
| CommitSession( ) | Method is called to commit the updates to permanent storage on the member server. |
| ReadyToCommitSession( ) | A method that is called on the master & member server drivers before calling CommitSession( ) method on the member servers. Enables the drivers on the member servers to notify the engine of any errors that might prevent full deployment. |
| AbortSession( ) | Aborts the current session, e.g. releases all queued Update lists. |
| ReadStoreProperty( ) | Reads a store property. |
| WriteStoreProperty( ) | Writes a store property. |

IReplicationEngine

This interface is implemented by each replication engine. This API includes all notifications from drivers to engine and also exposes replication context (e.g. storage location during backup) to the engine. This API is also used for engine-to-engine communication (between the master server and the member servers). This interface includes the following interface methods:

| Method | Description |
|---|---|
| ListNotify( ) | Callback used by the driver. The driver uses this method to notify the engine with the various type of list that it has developed, e.g. Action lists are presented to ProcessActionList( ) on the master server, Update lists are presented to each member server using ProcessUpdateList( ) |
| DefinePath( ) | Analogous to IReplicationDriver interface. |
| SetServerList( ) | Sets the server list to use as the target for replication. |
| BeginSession( ) | Called by the master server engine on the member server engine. Begins a new session. |
| CommitSession( ) | Called by the master server engine to the member server engines. Commits the current session and returns only when replicated data has been committed to permanent storage. |
| ReadyToCommitSession( ) | Forwarded by the master server engine to the member server engines. Calls ReadyToCommitSession( ) on all drivers on both master and member drivers. This will take place only after ensuring that each driver during the replication session has generated its last Update list. |
| AbortSession( ) | Forwarded by the master server engine to the member server engines. Aborts the current session. |
| GetReplicationInfo( ) | Used to retrieve various information, e.g. roots of the backup storage area in a file system, identification of member server's replication engine, identification of replication ID for a particular replication session, etc. |
| TransferList( ) | Called by the master server engine on the member server engine and by member |

-continued

| Method | Description |
|---|---|
| | servers on the master server engine. Requests processing of a replication list (IHave, Action, Update). |

IReplicationControl

This interface manages replication jobs. The interface can be accessed using a command line utility. This utility can take its parameter from either the command line or a file. The interface includes the following methods:

| Method | Description |
|---|---|
| DefineReplication( ) | This method defines a replication. It can be called to designate automatic updates, to replicate a farm configuration, to include content defined in the system replication definition, to designate a full synch update, to prevent ACLs (Access Control Lists) from being translated, to override replication, and the like. |
| DeleteReplication( ) | Deletes an existing replication. |
| EditReplication( ) | Allows any of the parameters that constitute a replication definition to be updated. |
| EnumReplications( ) | Enumerates all defined replications. |
| GetReplicationInfo( ) | Retrieves information that has been already defined, e.g. defined in DefineReplication( ). |
| CreateReplicationJob( ) | This method will request an immediate replication by creating a replication job based on a replication definition. |
| AbortReplicationJob( ) | Aborts an in-progress Replication Job. |
| EnumReplicationJob( ) | Enumerates replication jobs. |
| GetReplicationJobInfo( ) | Retrieves replication job information. |
| GetReplicationJobStatus( ) | Returns the status for a specific replication job. There are two levels that return various degrees of information, e.g. level 1 returns the job state and status; level 2 adds a XML-formatted buffer filled with various information including total count of object to replicate and how many have been replicated thus far, object size, how much of the object has been transferred etc. |
| Backup( ) | Initiates a full synch to a file path to enable backup. |
| EnumBackup( ) | Enumerates all backups on a particular computer. |
| GetBackupInfo( ) | Retrieves backup information. |
| Restore( ) | Restores a backup on a particular computer. |
| DeleteBackup( ) | Removes a backup on a particular computer. |
| NotifyStateChange( ) | This method is called to signal a new member server state change to the master server. |

-continued

| Method | Description |
|---|---|
| AbortAllReplicationJobs( ) | Called on the master server and aborts all currently running replication sessions, including automatic sessions. |
| WaitForAllReplicationJobsCompletion( ) | Called on the master server. Returns when all current replication sessions are completed or aborted. |
| StopReplication( ) | Called on the master server and disables new replication sessions to be scheduled. |
| StartReplication( ) | Called on the master server to start replication session scheduling and automatic sessions. |
| EnumDriver( ) | Enumerates all of the drivers on a computer. |
| GetDriverAttr( ) | Retrieves the value of a named attribute for a specific driver. |
| SetDriverAttr( ) | Sets the value of a named attribute for a specific driver. |
| GetReplAttr( ) | Retrieves the value of a named attribute. |
| SetReplAttr( ) | Sets the value of a named attribute. |

In Operation

In operation, the described embodiment provides a convenient, easy to implement way of maintaining synchronization across multiple computing devices. In the described example, the computing devices constitute web servers although it should be apparent that any suitable computing devices can be synchronized in accordance with the described embodiment. The described embodiments provide a way to synchronize resource types and content across multiple different computing devices.

FIG. **5** shows a flow diagram that described steps in a method in accordance with the described embodiment. Aspects of the illustrated method are implemented on both the master server side and the member server side. Although the method can be implemented in connection with any suitable hardware, software, firmware or combination thereof, the method can be advantageously implemented in connection with the architecture shown in FIG. **4**.

FIG. **5** shows processing that takes place in an exemplary "full synch" mode. Step **500** generates a set of lists that describes a configuration of resources that each of a plurality of computing devices should have in order to be synchronized with one another. In the illustrated example, a master server generates the list through the use of a plurality of resource drivers each of which is associated with a different resource. Each resource driver is responsible for generating their own individual lists that describes their resources and sending the lists to a master replication engine. The replication engine then assimilates all of the lists that are received from the various master server drivers. Step **502** sends the lists (IHave lists) that are prepared by the master server replication engine to each of a number of different member servers. When the member servers receive the lists, each member server evaluates the lists (step **504**) to determine whether their resource configuration is synchronized with that of the master server. In the illustrated example, each member server has a software architecture that mirrors the architecture of the master server. Accordingly, the lists are received by the member server's replication engine and

individual list are sent to the corresponding member server drivers for evaluation. If, as a result of the evaluation that takes place at the member server, it is determined that one or more resources are not synchronized with a corresponding resource on the master server, step **506** generates a response to the master server. In this example, each member server driver is responsible for their own individual resource evaluation as well as generating a portion of a response that is to be sent to the master server. The member server drivers send their lists to the member server's replication engine which assimilates all of the lists that are generated by the individual drivers. The member server's replication engine then sends the response (Action lists) to the master server (step **508**). The master server receives the response from the member server (step **510**) and evaluates it (step **512**) to determine the resource needs of a particular member server. In the illustrated example, individual master server drivers are handed lists that pertain to the resource with which they are associated. The drivers are then responsible for obtaining the content that is necessary to bring the corresponding member server resource into synchronization with the master server resource. This content or data is then sent to the member server (step **514**) for use in synchronizing its associated resource. In the described embodiment, content or data can either be sent separately, e.g. files sent through a file-sending protocol (such as HTTP or FTP) or the data can be embedded in the lists (Update lists) that are returned to the member server. The member server then receives the content or data that is sent from the master server (step **516**) and uses the data to synchronize its configuration of resources.

FIG. **6** shows steps in a method in accordance with an "automatic mode" in which the master server is monitored r any changes in resources. Step **600** monitors the resources that are resident on the master server. In the illustrated example, each of the master server drivers can be configured to monitor their associated resource. Step **602** determines whether there are any changes to the resource configuration. Here, if one of the drivers detects a change, it can notify the master server's replication engine, which then handles it as for the "full-synch" mode. In this example, each or any of the drivers can generate the Update list. The Update list is received by the member servers and used to update their resource configuration (step **606**).

The inventive methods and systems described above provide a simple, flexible approach to maintaining a configuration of resources synchronized across multiple computing devices. The methods and systems are flexible because many different types of resources, e.g. not just simply file resources, can be maintained in a synchronized state at any one time. In the illustrated example, economies are achieved by having each resource on a computing device associated with a different driver that is programmed to maintain its resource. The driver then ensures that its associated resource is synchronized with other resources that are present on other computing devices. Changes to resource configurations are passed ultimately to the associated driver which then ensures that the changes are promulgated to the resource for which the driver is responsible. Advantageously, the inventive methods and systems can be used in the context of web servers, e.g. web server clusters, to ensure that configurations and hence functionalities are standard across multiple different servers.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

What is claimed is:

1. A computer network comprising:

a master computing device comprising a configuration of resources, the master computing device being configured to develop a set of lists that describes aspects of its configuration of resources; and

a plurality of additional computing devices communicatively linked with the master computing device, each additional computing device comprising a configuration of resources that correspond, at least in part, to the configuration of resources of the master computing device, each additional computing device being configured to receive the lists that are developed by the master computing device and evaluate the lists to determine whether the particular configuration of resources corresponding to the respective computing device is synchronized with those of the master computing device,

each additional computing device, responsive to evaluating the lists, is configured to:

prepare a second set of lists that describes the resource configuration needs corresponding to the respective computing device in the event that the additional computing device is not synchronized with the master computing device; and

send the second set of lists to the master computing device.

2. The computer network of claim 1, wherein at least one of the computing devices comprises a web server.

3. The computer network of claim 1, wherein a plurality of the computing devices comprise web servers.

4. The computer network of claim 1, wherein all of the computing devices comprise web servers.

5. The computer network of claim 1, wherein the master computing device is further configured to:

receive the second set of lists from each of the additional computing devices;

evaluate the second set of lists to determine the resource configuration needs for, a corresponding additional computing device; and

send data to the corresponding additional computing device that can be used to synchronize the resource configurations for the corresponding additional computing device.

6. The computer network of claim 1, wherein all of the computing devices are configured to implement a common functionality.

7. The computer network of claim 1, wherein the lists comprise extensible markup language (XML) lists.

8. A computer network comprising:

a master server comprising a configuration of resources that define the content and settings for the master server, the master server being configured to develop a set of lists that describes aspects of its configuration of resources; and

a plurality of member servers communicatively linked with the master server, each member server having its own configuration of resources that correspond, at least in part, with those of the master server, each member server being configured to:

receive the set of lists that is developed by the master server;

evaluate the set of lists to determine whether its particular configuration of resources are synchronized with those of the master server; and

communicate to the master server any resource needs associated with an individual resource of its configuration of resources in the event the configuration of resources is found to be unsynchronized with the corresponding configuration of resources of the master server.

9. The computer network of claim 8, wherein all of the servers are web servers.

10. The computer network of claim 8, wherein each of the member servers comprises a plurality of drivers, each driver being associated with a different resource type of the member server's configuration of resources, each driver being configured to:

evaluate a set of lists that pertains to its associated resource to determine whether the resource is synchronized with a corresponding resource of the master server; and

prepare a set of lists that describes resource needs that will synchronize the associated resource in the event that the associated resource is not synchronized with the corresponding resource of the master server.

11. The computer network of claim 10, wherein each driver is further configured to:

receive data from the master server for updating its associated resource; and

use the data that is received from the master server to update its associated resource.

12. The computer network of claim 8, wherein said aspects of the resource configurations that are described in the lists that are developed by the master server comprise at least one signature that represents a particular resource of the configuration of resources.

13. The computer network of claim 8, wherein each member server is configured to communicate to the master server by building a set of lists that describes its resource needs and communicating the set of lists to the master server.

14. The computer network of claim 13, wherein the master server is configured to:

receive the set of lists that is built by each member server,

evaluate the set of lists that is received from each member server; and

send data to each of the member servers that enables the member servers to synchronize one or more resources that are not synchronized with those of the master server.

15. The computer network of claim 8, wherein the lists comprise extensible markup language (XML) lists.

16. A computing device comprising:

a master server comprising a configuration of resources that define the content and settings for the master server, the master server being configured to:

develop a set of lists that describes aspects of its configuration of resources;

send the set of lists to a plurality of member servers;

receive communications from the individual member servers responsive to sending the set of lists, the communications being capable of requesting data from the master server that enable the member servers to synchronize their configuration of resources with those of the master server; and

send data to a member server that can be used by the member server to synchronizers its configuration of resources.

17. The computing device of claim 16, wherein the master server comprises a web server.

18. The computing device of claim 16, wherein the master server and each of the member servers comprise a web server.

19. The computing device of claim 16, wherein the data that is sent to the member server is embedded in a list.

20. The computing device of claim 16, wherein the master server comprises a plurality of drivers, each driver being associated with a different resource type of the master server's configuration of resources, each driver being configured to prepare a portion of the list that pertains to its associated; resource.

21. The computing device of claim 16, wherein the lists comprise extensible markup language (XML) lists.

22. A computing device comprising:

one, or more server computers each of which comprising a configuration of resources tat define its content and settings, each server computer being configured to:

evaluate its own resources and request list of data that defines at least in part the configuration of resources that the server computer should have;

receive a set of update lists from a master server; and

update its configuration of resources using the lists that are received from the master server.

23. The computing device of claim 22, wherein the server computers comprise web servers.

24. The computing device of claim 22, wherein each server computer comprises a plurality of drivers, each driver being associated with a different resource type of the server computer's configuration of resources, each driver being configured to receive an update list that corresponds to its associated resource and update the resource configuration using the update list.

25. The computing device of claim 22, wherein the request lists and update lists comprise extensible markup language (XML) lists.

26. A computer-implemented method of synchronizing a configuration of resources on a plurality of computing devices comprising:

generating a set of lists that describes a configuration of resources that each of a plurality of computing devices should have in order to be synchronized with one another, the configuration of resources defining the content and the settings for each of the computing devices;

sending the set of lists to each of the computing devices;

receiving a response from one or more of the computing devices, each response requesting data that is needed in order to synchronize the configuration of resources for the corresponding computing device;

evaluating the response to determine what data is needed by a particular computing device to synchronize its resources; and

sending the data that is needed by the particular computing device to the computing device so that it can synchronize its resources.

27. The method of claim 26, wherein said computing devices comprise web servers.

28. The method of claim 27, wherein said generating of the set of lists comprises generating the set of lists using a master computing device composing a configuration of resources that are desired to be synchronized across said plurality of computing devices.

29. The method of claim 28, wherein the master computing device and the plurality of computing devices comprise web servers.

30. The method of claim 26, further comprising:

taking a signature of one or more resources of the configuration of resources; and

incorporating the signature into the set of lists.

31. The method of claim 26, wherein said sending of the data comprises embedding at least some of the data in the set of lists.

32. The method of claim 26, wherein the lists comprise extensible markup language (XML) lists.

33. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, implement the method of claim 26.

34. One or more programmable web servers, each web server comprising;

one or more processors;

memory associated with the one or more processors; and

software code stored in the memory which, when executed by the one or more processors, implements the method of claim 26.

35. A computer-implemented method of synchronizing a configuration of resources on a computing device comprising:

receiving a set of lists that describes a configuration of resources that a first computing device should have in order to be synchronized with a configuration of resources of a second computing device, the configurations of resources defining the content and settings for the first and second computing devices;

evaluating the set of lists to determine whether the first computing device's configuration of resources is synchronized with the configuration of resources of the second computing device;

responsive to said evaluating and in the event that the first computing device's configuration of resources is not synchronized with the configuration of resources; of the second computing device, generating a response that contains information about resources that are needed for the first computing device in order to synchronize it with the configuration of resources of the second computing device;

sending the response to the second computing device;

responsive to said sending, receiving data from the second computing device that is to be used by the first computing device in order to synchronize it with the second computing device; and

synchronizing the first computing device using the data that was received from the second computing device.

36. The method of claim 35, wherein the first and second computing devices comprise web servers.

37. The method of claim 35, wherein the first and second computing devices comprise computing devices that are configured to implement a common functionality.

38. The method of claim 35, wherein the first and second computing devices comprise web servers that are configured to implement a common functionality.

39. The method of claim 35, wherein the first computing device comprises a plurality of drivers, each driver being associated with a different one of the resources of the configuration of resources, wherein:

said evaluating comprises, for at least one of the drivers, using said at least one driver to evaluate only a portion of the set of lists that pertains to the resource with which the driver is associated; and

said generating comprises generating at least a portion of the response using said at least one driver.

40. The method of claim 35, wherein said receiving of the data comprises receiving a set of lists, at least some of the data being embedded in the set of lists.

41. The method of claim 35, wherein the lists comprise extensible markup language (XML) lists.

42. One or more computer-readable media having computer-readable instructions thereon which, when executed by a computer, implement the method of claim 35.

43. A computer-implemented method of synchronizing resource configurations on a plurality of computing devices comprising:

generating a set of lists tat describes a configuration of resources that each of a plurality of computing devices should have in order to be synchronized with a master computing device, the configuration of resources defining the content and the settings for each of the computing devices;

sending the set of lists to each of the computing devices;

evaluating the set of lists at each of the computing devices to determine whether any resources of the configuration of resources of the individual computing devices are synchronized with the resource configurations of the master computing device;

if any of the individual computing devices are not synchronized with the master computing device:

generating a response for each computing device that describes data that is needed for a particular computing device in order for it to synchronize its configuration of resources with the configuration of resources of the master computing device;

sending the response to the master computing device;

evaluating the response at the master computing device to determine what configuration of resources is needed by the particular computing device that sent the response;

sending data that is needed by the particular computing device to the particular computing device so that it can synchronize itself; and

using the data that is sent, synchronizing the particular computing device.

44. The method of claim 43, wherein all of the computing devices are web servers.

45. The method of claim 43, wherein the lists comprise extensible markup language (XML) lists.

46. The method of claim 45, wherein said generating of the response comprises generating an extensible markup language (XML) response.

47. One or more computer-readable media having computer-executable instructions thereon which, when executed by one or more computers, implements the method of claim 43.

48. A server software architecture embodied on one or more computer-readable media for implementation on a computer server system, the server software architecture comprising:

a replication engine to handle communication between a master server and a plurality of member servers; and

at least one driver capable of communicating with the replication engine, the driver being associated with a particular resource that is resident on the master server, the driver being configured to:

prepare a list that describes characteristics of its associated resource; and

articulate the list to the replication engine;

the replication engine being configured to receive the list from the one driver, send the list to one or more

of the member servers so that the member servers can use the list to synchronize a resource that corresponds to the resource with which the one driver is associated, and receive data from one or more of the member services describing resource needs of the particular member server in the event that a resource is determined not to be synchronized with the corresponding resource on the master server.

**49**. The server software architecture of claim **48**, wherein the master server and the members servers comprise web servers.

**50**. The server software architecture of claim **48**, wherein the master server and the members servers comprise web servers that are individually configured to implement a common functionality.

**51**. The server software architecture of claim **48**, wherein the driver is configured to make at least one signature pertaining to a resource with which it is associated.

**52**. The server software architecture of claim **48** wherein said at least one driver comprises a plurality of drivers each of which being associated with a different resource that is resident on the master server.

**53**. The server software architecture of claim **48**, wherein the list comprises an extensible markup language (XML) list.

**54**. A server software architecture embodied on one or more computer-readable media for implementation on a computer server system, the server software architecture comprising:

a first replication engine associated with a master server, the first replication engine handling communication between the master server and a plurality of member servers;

a plurality of master server drivers capable of communicating with the first replication engine, each master server driver being associated with a different; resource that is resident on the master server and being configured to:

prepare a list that describes characteristics of its associated resource; and

articulate the list to the first replication engine; the first replication engine being configured to receive a list from each of the master server drivers and send one or more lists to one or more of the member servers so that the member servers can use the one or more lists to synchronize one or more resources that correspond to resources that are resident on the master server;

a plurality of second replication engines each of which being associated with different ones of the plurality of member savers, the second replication engines handling communication between their associated member server and the master server and being configured to receive one or more lists that might be sent by the first replication engine; and

a collection of member server drivers associated with each of the member servers, each of the member server drivers of a collection being associated with a different resource that is resident on the associated member server, each member server driver being configured to:

evaluate the resource with which it is associated, in response to its second replication engine receiving a list from the first replication engine;

determine whether the resource is synchronized with a corresponding resource on the master server; and

prepare a response to the master server that describes resource needs of the member server in the event that a resource is determined not to be synchronized with the corresponding resource on the master server.

**55**. The server software architecture of claim **54**, wherein the master server and the member servers are web servers.

**56**. The server software architecture of claim **54**, wherein the list comprises an extensible markup language (XML) list.

* * * * *

US006727917B1

(12) **United States Patent**
**Chew et al.**

(10) **Patent No.: US 6,727,917 B1**
(45) **Date of Patent: Apr. 27, 2004**

(54) **USER INTERFACE FOR PALM-SIZED COMPUTING DEVICES AND METHOD AND APPARATUS FOR DISPLAYING THE SAME**

(75) Inventors: **Chee H. Chew**, Redmond, WA (US); **Elizabeth A. Bastiaanse**, Cambridge, MA (US); **Jeffrey R. Blum**, Seattle, WA (US); **Christen E. Coomer**, Seattle, WA (US); **Mark H. Enomoto**, Seattle, WA (US); **Greg A. Keyser**, Sammamish, WA (US); **Kathryn L. Parker**, Fall City, WA (US); **William H. Vong**, Redmond, WA (US); **Sarah E. Zuberec**, Seattle, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/564,928**

(22) Filed: **May 4, 2000**

**Related U.S. Application Data**

(60) Provisional application No. 60/175,112, filed on Jan. 6, 2000.

(51) **Int. Cl.**[7] ..... **G06F 3/14**
(52) **U.S. Cl.** ..... **345/765**; 345/179
(58) **Field of Search** ..... 345/701, 776, 345/808, 157, 158, 169, 173, 764, 765, 835, 179

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 5,588,105 | A * | 12/1996 | Foster et al. | 345/700 |
| 5,666,438 | A * | 9/1997 | Beernink et al. | 382/189 |
| 6,243,071 | B1 * | 6/2001 | Shwarts et al. | 345/146 |

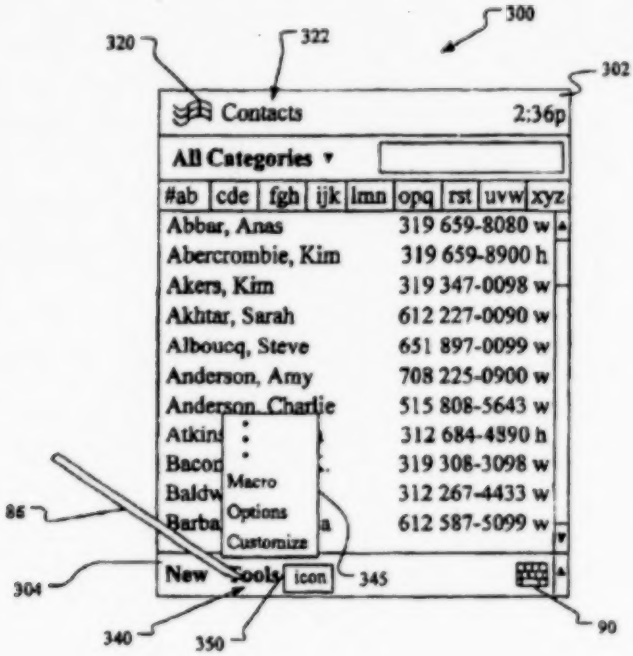
* cited by examiner

*Primary Examiner*—Cao (Kevin) Nguyen
(74) *Attorney, Agent, or Firm*—Westman, Champlin & Kelly, P.A.

(57) **ABSTRACT**

The hand-held computing device user interface (UI) of the present invention displays information for an active application program in a middle portion of the screen, and displays a shell program controlled navigation bar at a top portion of the screen. The navigation bar includes a navigation icon which when tapped by the stylus aids the user in navigating to other application programs. The navigation bar also includes a title for the active application program, thus saving vertical real estate on the screen. The UI of the present invention also displays an application menu bar at a bottom portion of the screen so that the user of the hand-held device can manipulate data from the active application by tapping menu items with a stylus without blocking view of the middle portion of the display.

**20 Claims, 8 Drawing Sheets**



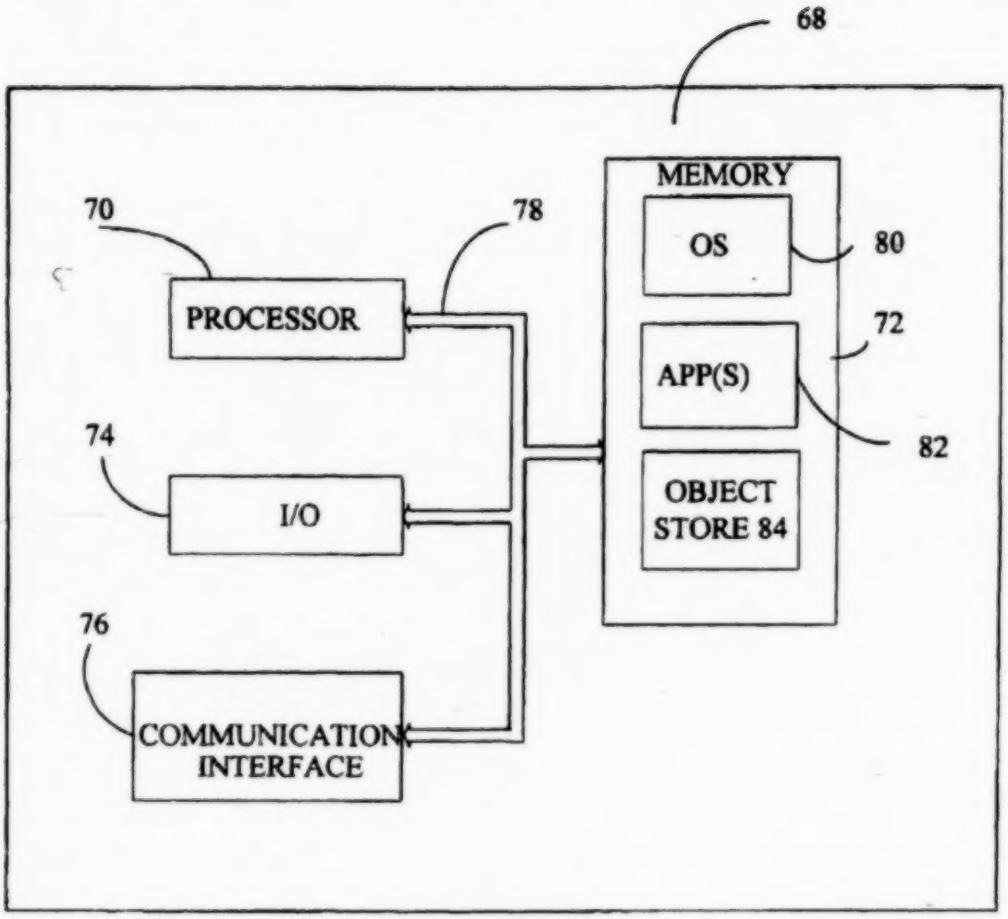

68
MEMORY
70
78
OS
80
PROCESSOR
72
APP(S)
82
74
I/O
OBJECT STORE 84
76
COMMUNICATION INTERFACE

FIG. 1

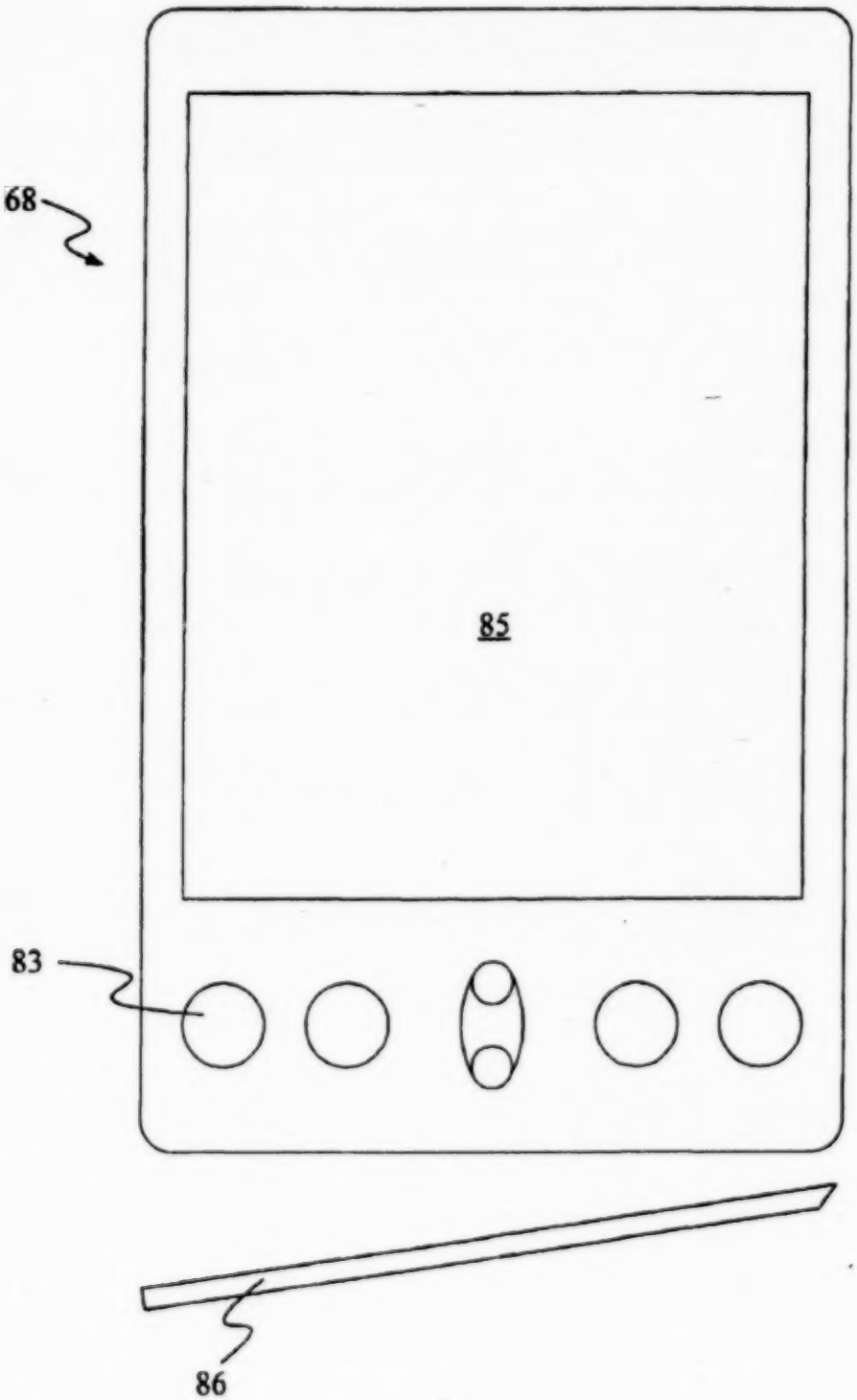



FIG. 2

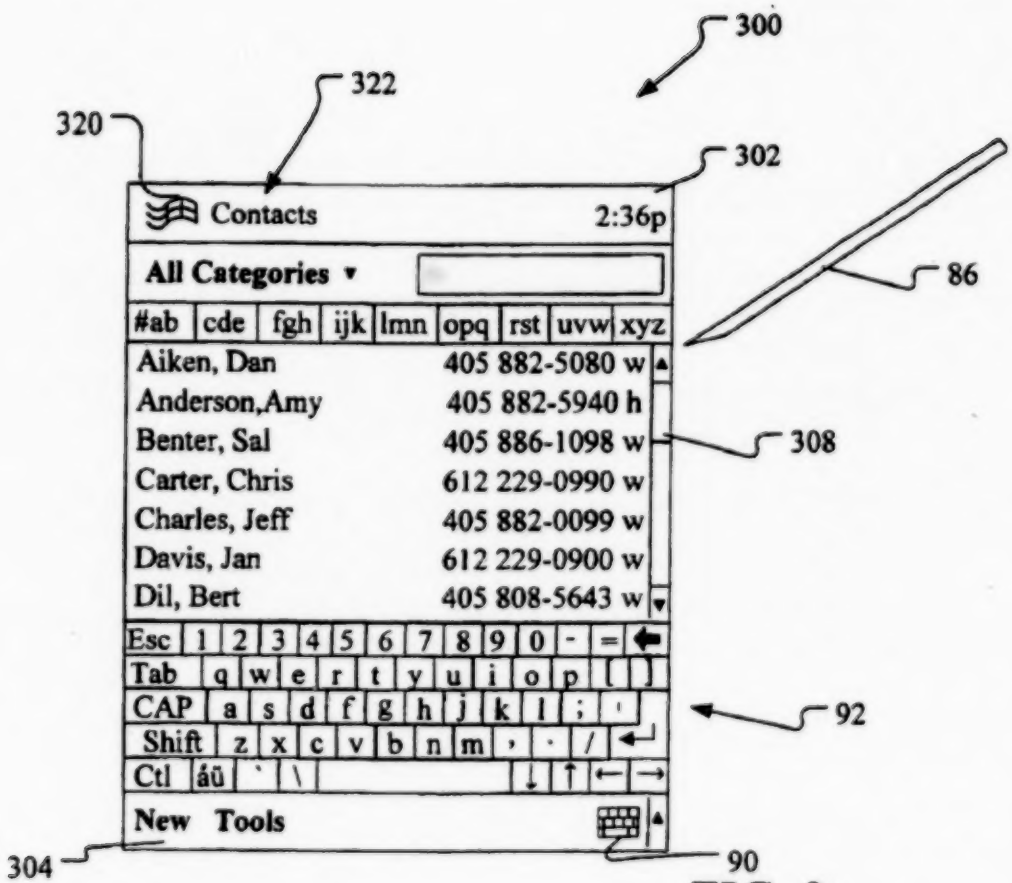

300
322
320
302
Contacts
2:36p
All Categories
86
#ab cde fgh ijk lmn opq rst uvw xyz
Aiken, Dan 405 882-5080 w
Anderson,Amy 405 882-5940 h
Benter, Sal 405 886-1098 w
Carter, Chris 612 229-0990 w
Charles, Jeff 405 882-0099 w
Davis, Jan 612 229-0900 w
Dil, Bert 405 808-5643 w
308
Esc 1 2 3 4 5 6 7 8 9 0 - = ⬅
Tab q w e r t y u i o p [ ]
CAP a s d f g h j k l ; '
Shift z x c v b n m , . / ↵
Ctl áü ` \ ↓ ↑ ← →
92
New Tools
304
90

FIG. 3

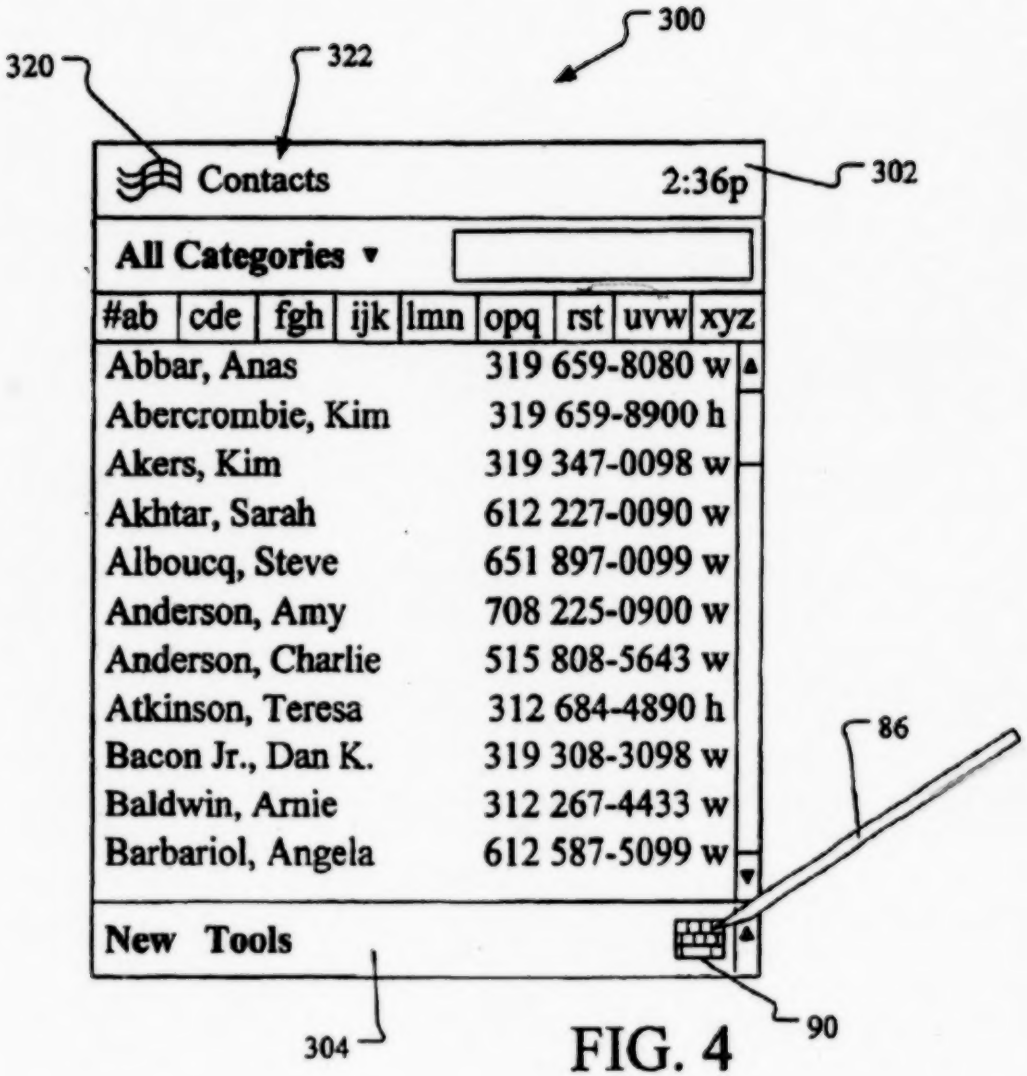

300
320
322
Contacts
2:36p
302
All Categories
#ab cde fgh ijk lmn opq rst uvw xyz
Abbar, Anas 319 659-8080 w
Abercrombie, Kim 319 659-8900 h
Akers, Kim 319 347-0098 w
Akhtar, Sarah 612 227-0090 w
Albouq, Steve 651 897-0099 w
Anderson, Amy 708 225-0900 w
Anderson, Charlie 515 808-5643 w
Atkinson, Teresa 312 684-4890 h
Bacon Jr., Dan K. 319 308-3098 w
Baldwin, Arnie 312 267-4433 w
Barbariol, Angela 612 587-5099 w
86
New Tools
304
90

FIG. 4

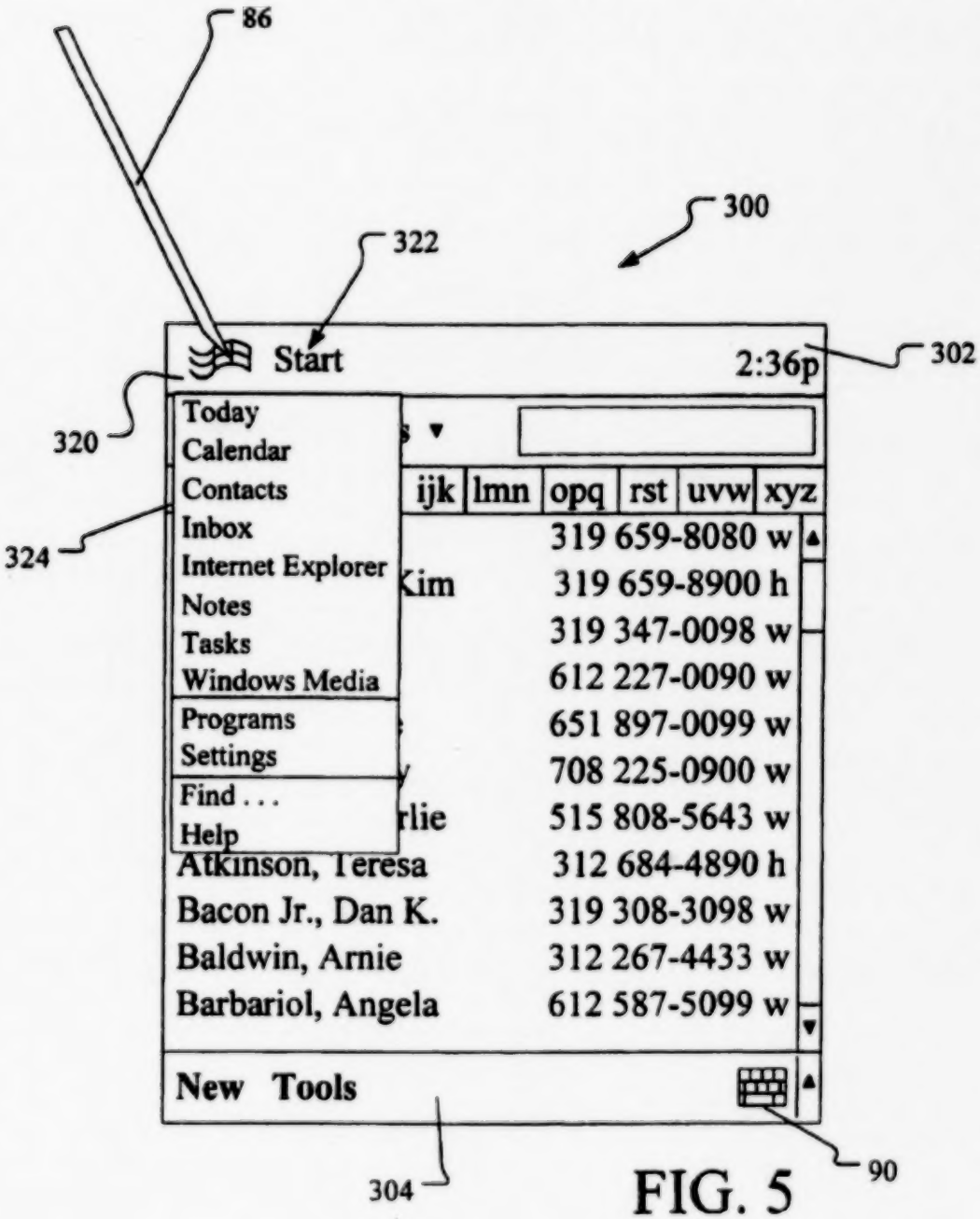

86
300
322
Start
2:36p
302
320
Today
Calendar
Contacts
Inbox
Internet Explorer
Notes
Tasks
Windows Media
Programs
Settings
Find . . .
Help
324
ijk lmn opq rst uvw xyz
319 659-8080 w
319 659-8900 h
319 347-0098 w
612 227-0090 w
651 897-0099 w
708 225-0900 w
515 808-5643 w
312 684-4890 h
319 308-3098 w
312 267-4433 w
612 587-5099 w
Atkinson, Teresa
Bacon Jr., Dan K.
Baldwin, Arnie
Barbariol, Angela
New Tools
304
90

FIG. 5

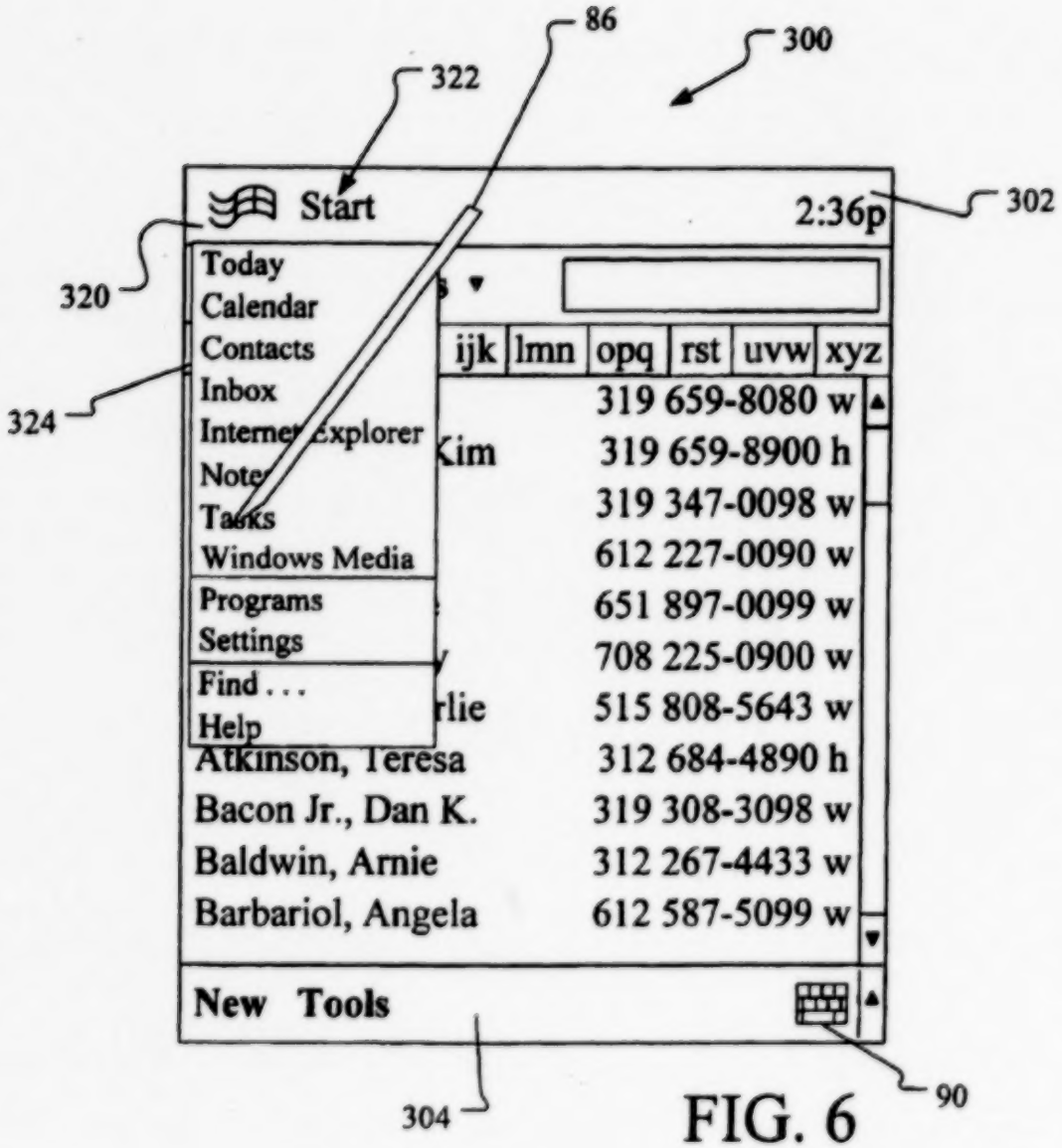

86
300
322
Start
2:36p
302
320
Today
Calendar
Contacts
Inbox
Internet Explorer
Notes
Tasks
Windows Media
Programs
Settings
Find . . .
Help
324
ijk lmn opq rst uvw xyz
319 659-8080 w
319 659-8900 h
319 347-0098 w
612 227-0090 w
651 897-0099 w
708 225-0900 w
515 808-5643 w
312 684-4890 h
319 308-3098 w
312 267-4433 w
612 587-5099 w
Atkinson, Teresa
Bacon Jr., Dan K.
Baldwin, Arnie
Barbariol, Angela
New Tools
304
90

FIG. 6

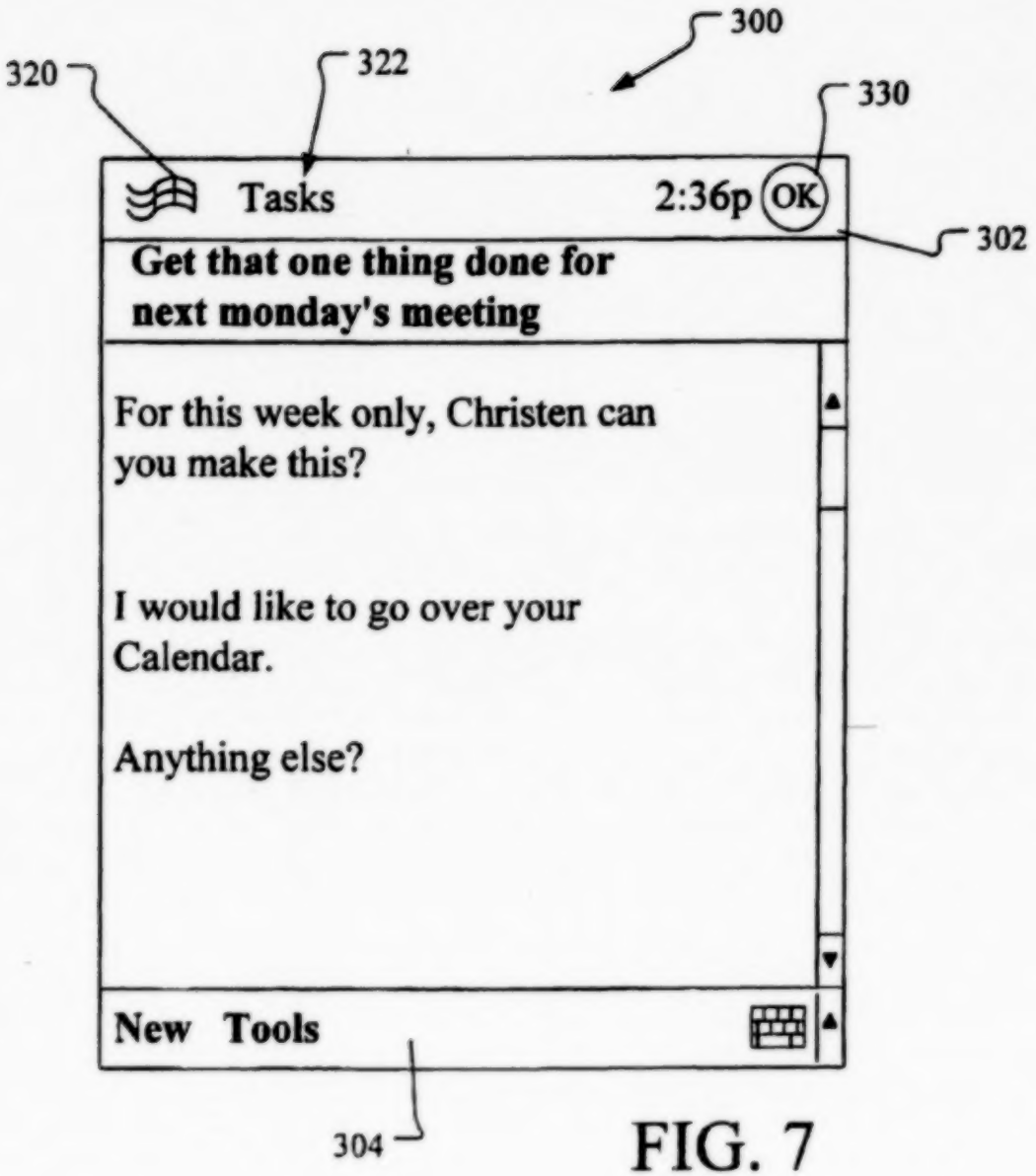

300
320
322
330
Tasks
2:36p
OK
302
Get that one thing done for next monday's meeting
For this week only, Christen can you make this?
I would like to go over your Calendar.
Anything else?
New Tools
304

FIG. 7

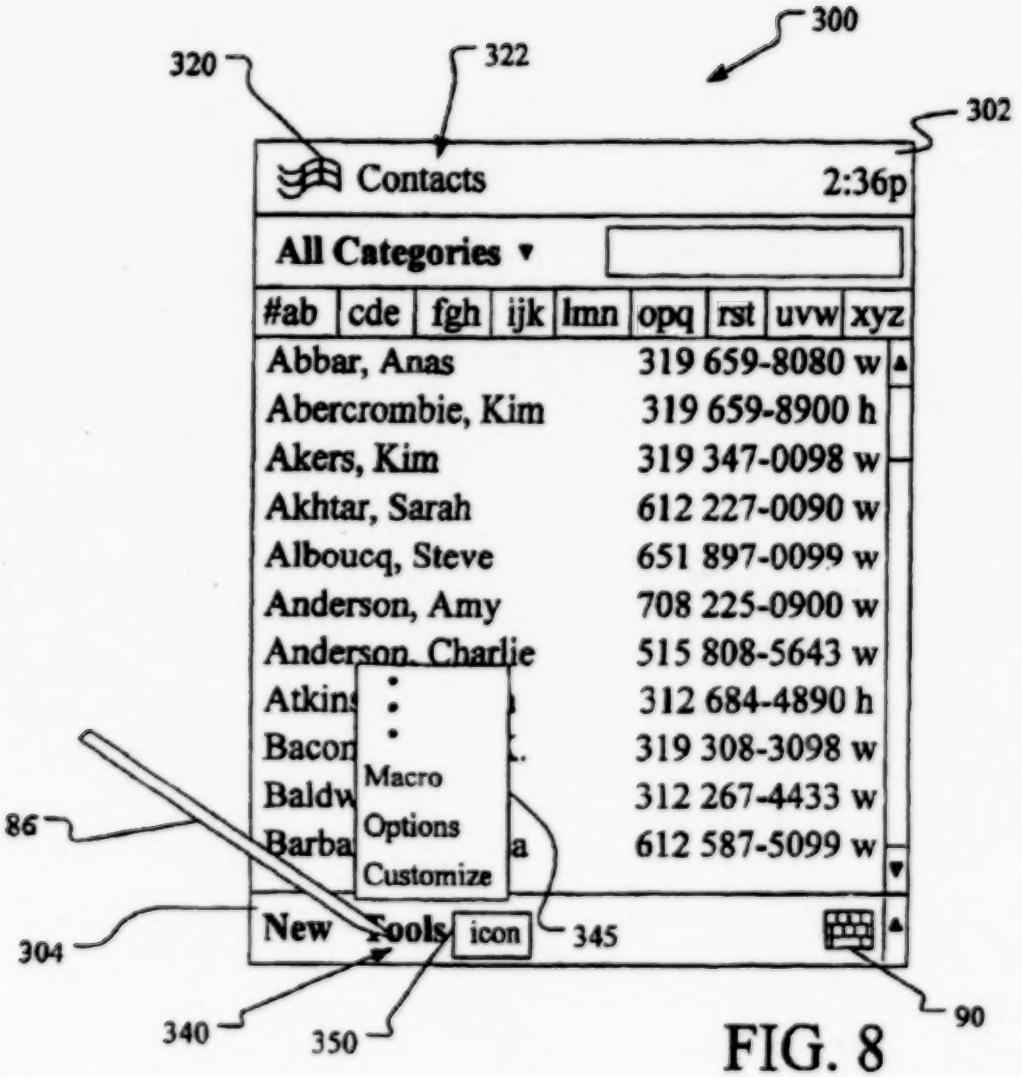

300
320
322
302
Contacts
2:36p
All Categories
#ab
cde
fgh
ijk
lmn
opq
rst
uvw
xyz
Abbar, Anas 319 659-8080 w
Abercrombie, Kim 319 659-8900 h
Akers, Kim 319 347-0098 w
Akhtar, Sarah 612 227-0090 w
Alboucq, Steve 651 897-0099 w
Anderson, Amy 708 225-0900 w
Anderson, Charlie 515 808-5643 w
312 684-4890 h
319 308-3098 w
312 267-4433 w
612 587-5099 w
Macro
Options
Customize
86
304
New
Tools
icon
345
340
350
90

FIG. 8

# USER INTERFACE FOR PALM-SIZED COMPUTING DEVICES AND METHOD AND APPARATUS FOR DISPLAYING THE SAME

## REFERENCE TO RELATED APPLICATION

The present application claims priority from U.S. Provisional Application Ser. No. 60/175,112, filed on Jan. 6, 2000 and entitled "USER INTERFACE FOR PALM-SIZED COMPUTING DEVICES AND METHOD AND APPARATUS FOR DISPLAYING THE SAME."

## REFERENCE TO CO-PENDING APPLICATIONS

Reference is hereby made to co-pending U.S. patent applications, entitled "METHOD AND APPARATUS FOR FAST SEARCHING OF HAND-HELD CONTACTS LISTS", attorney docket number M61.12-0225; entitled "SMART COLUMN FOR CONTACT INFORMATION ON PALM-SIZED COMPUTING DEVICES AND METHOD AND APPARATUS FOR DISPLAYING THE SAME", attorney docket number M61.12-0226; entitled "USER INTERFACE FOR PALM-SIZED COMPUTING DEVICES AND METHOD AND APPARATUS FOR DISPLAYING THE SAME", attorney docket number M61.12-0227; entitled "METHOD AND APPARATUS FOR PROVIDING CONTEXT MENUS ON A PEN-BASED DEVICE", attorney docket number M61.12-0228; entitled "METHOD AND APPARATUS FOR PROVIDING RECENT CATEGORIES ON A HAND-HELD DEVICE", attorney docket number M61.12-0229.

## BACKGROUND OF THE INVENTION

The present invention relates to hand-held personal computing devices commonly known as hand-held devices or palm-size devices. More particularly, the present invention relates to an improved user interface (UI) for a palm-held portable computer.

Hand-held devices are small electronic computing devices, sometimes also referred to as personal digital assistants (PDAs). Generally, these devices fit comfortably in the palm of a user's hand. Hand-held devices have gained a niche as personal organizers, allowing users to view or change lists of contacts, tasks or calendar events on a display using an interface device such as a keyboard (typically integrated into a touch sensitive display) and a stylus.

In a hand-held device running more than one application, there is a need for improved UI mechanisms for manipulating data, for providing feedback, for controlling the applications, and for navigating between the different applications. All these peripheral UI mechanisms are needed around the actual data content of the applications. However, there is very limited screen or display real estate available for displaying both the data and the UI for the applications and/or shell program. As screen real estate is utilized for UI application functions, the amount of data which can be displayed to the user decreases accordingly. Thus, there is a need for methods of efficiently displaying application and shell program information on a limited size screen.

## SUMMARY OF THE INVENTION

The present invention includes apparatus and methods for generating a user interface (UI), for a stylus based hand-held device, which maximizes screen real estate available for displaying data, and which provides a mechanism to efficiently manipulate data and to switch between application programs.



The hand-held computing device UI of the present invention displays information for an active application program in a middle portion of the screen, and displays a shell program controlled navigation bar at a top portion of the screen. The navigation bar includes a navigation icon which when tapped by the stylus aids the user in navigating to other application programs. The navigation bar also includes a title for the active application program, thus saving vertical real estate on the screen. The UI of the present invention also displays an application menu bar at a bottom portion of the screen so that the user of the hand-held device can manipulate data from the active application by tapping menu items with a stylus without blocking the view of the middle portion of the display.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a hand-held device computing environment in which embodiments of the present invention may be practiced.

FIG. **2** is a front view of a hand-held computing device on which embodiments of the present invention may be practiced.

FIG. **3** is a screen-shot of a display showing a portion of a hand-held device user interface (UI) in accordance with embodiments of the invention and showing a soft input panel.

FIG. **4** is a screen-shot of a display showing a portion of the hand-held device UI shown in FIG. **3**, showing the soft input panel removed from view.

FIG. **5** is a screen-shot of a display showing the portion of the UI of FIG. **4**, but with a shell program navigation or start menu deployed.

FIGS. **6–7** are screen-shots of the display shown in FIG. **5**, which illustrate UI mechanisms for changing application information displayed by the shell program.

FIG. **8** is a screen-shot of a display showing a portion of the UI of the invention with an application tool menu deployed.

## DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

FIG. **1** is a block diagram of a hand-held device **68**, which is an exemplary computing environment for the present invention. Hand-held device **68** includes a microprocessor **70**, memory **72**, input/output (I/O) components **74**, a communication interface **76** for communicating with, for example, a separate computer or hand-held device. In one embodiment, the aforementioned components are coupled for communication with one another over a suitable bus **78**.

Memory **72** is implemented as non-volatile electronic memory such as random access memory (RAM) with a battery back-up module (not shown) such that information stored in memory **72** is not lost when the general power to mobile device **68** is shut down. A portion of memory **72** is preferably allocated as addressable memory for program execution, while another portion of memory **72** is preferably used for storage, such as to simulate storage on a disk drive.

Memory **72** includes an operating system **80**, application programs **82** as well as an object store **84**. During operation, operating system **80** is preferably executed by processor **70** from memory **72**. Operating system **80**, in one preferred embodiment, is a "WINDOWS CE" brand operating system commercially available from Microsoft Corporation, also known as Windows for Pocket PC. Operating system **80** is preferably designed for hand-held devices, and implements

database features that can be utilized by applications **82** through a set of exposed application programming interfaces and methods. The objects in object store **84** are maintained by applications **82** and operating system **80**, at least partially in response to calls to the exposed application programming interfaces and methods.

Input/output components **74** can include, for example, input devices such as a keyboard, a mouse or other pointing device, touch sensitive display components, and control buttons. Input/output components **74** can also include output devices such as liquid crystal displays and speakers. Further, input/output components **74** include electronics and/or software associated with these devices or with other devices such as printers.

Communication interface **76** represents numerous devices and technologies that allow hand-held device **68** to send and receive information. The devices include wired and wireless modems, satellite receivers and broadcast tuners to name a few. In some embodiments, mobile device **68** can also be directly connected to a computer to exchange data therewith. In such cases, communication interface **76** can be an infrared transceiver or a serial, parallel, USB, Ethernet or other type of communication connection, all of which are capable of transmitting streaming information. While communication interface **76** is included in various forms in many embodiments of the invention, communication interface **76** is optional in yet other embodiments if sending and receiving information is not required.

FIG. **2** is a simplified pictorial illustration of device **68**. Device **68** can be a desktop assistant sold under the designation H/PC; a palm-size PC or Pocket PC designated as P/PC having software provided by the Microsoft Corporation. In one embodiment, device **68** includes a set of control buttons **83**, display **85** and stylus **86**. In the embodiment shown in FIG. **2**, display **85** is a liquid crystal display (LCD) which uses a contact sensitive display screen in conjunction with stylus **86**. Stylus **86** is used to press or contact display **85** at designated coordinates to accomplish certain user input functions. In some embodiments, a miniaturized keyboard with alpha-numeric keys is provided with the mobile device. In other embodiments, a "soft" keyboard **92** accessible through a soft input panel (SIP) button/icon **90** provided through a contact sensitive display screen (shown in FIGS. **3–10**) on the device. In yet other embodiments, a character recognition module is employed to recognize characters written on the contact sensitive display screen using stylus **86**.

In some hand-held device UIs, a number of difficulties can be experienced. For example, in some instances the UI provides no title bar indicating the name of the current application program being displayed to the user. Generally, for optimal use, the user needs to be aware of which application is active and must allow time for his or her mind to discern that they are in the particular application (e.g., a Contacts application). However, if a conventional application program title bar were simply added to the UI, vertical screen real estate would be consumed, which is already in short supply in a small hand-held device.

Another characteristic of conventional hand-held device UIs is that the application commands (the commands that manipulate the data on the screen) are located at the top of the screen. As a result, for the user to select a data manipulating command or device, his or her hand frequency has to obscure the data in order to get to the commands with the stylus. Meanwhile, the shell navigation program icon, which is used to switch between application programs, is displayed at the bottom of the screen. Therefore, when the user wishes to navigate away from the current data set (i.e., wishes to utilize a different data set and application program), his or her hand does not obscure the screen.

A hand-held device UI of the present invention addresses these problems. FIGS. **3–8** illustrate features of a hand-held device UI **300** that is shown on display **85** of FIG. **2** under various embodiments of the present invention. In FIGS. **3–6** and **8**, UI **300** is shown in use with a Contacts application program. However, UI **300** is not limited to use with Contacts applications, and can be used with a wide variety of applications adapted for hand-held devices.

As shown in FIG. **3** with a Contacts window active, UI **300** includes a shell program navigation bar **302** and an application program menu bar **304**. UI **300** also includes scroll bar **308** for moving within the active application program, as well as a soft input panel (SIP) **92** and an SIP button/icon **90**. By tapping stylus **86** on the letters on SIP **92**, the user is able to input data into the application programs.

Referring to FIG. **4**, it can be seen that by tapping SIP icon **90** with stylus **86**, SIP **92** can be removed from view in the window of UI **300** to allow for additional screen real estate on which the shell and application programs may be displayed. FIGS. **4–8** illustrate UI **300** with SIP **92** removed from view.

Referring back to FIG. **3**, UI **300** is provided with several significant differences from prior art hand-held device UIs. As in the prior art, shell program navigation bar **302** can include navigation or start menu icon **320** for use in implementing shell program functions and for navigating between application programs (i.e., Contacts programs, Tasks programs, Calendar programs and the like). The start menu icon shown is a trademark of Microsoft Corporation, and other start or navigation menu icons can be used in the present invention. In contrast to prior art hand-held device UIs, in UI **300** navigation bar **302** is positioned at the top of the screen instead of at the bottom of the screen. Also in contrast to the prior art, this navigation bar serves a dual purpose.

First, navigation bar **302** serves as a shell program navigation tool in the same or similar manner as navigation or start menus in the prior art. The second purpose provided by shell program navigation bar **302** is that it also acts as a title bar for the active application program. Thus, it displays the title or name **322** of the current active application, which in FIG. **3** is a Contacts application. Application title **322** displayed by shell program navigation bar **302** closely resembles prior art application title bars so that the user quickly comprehends that this is the title of the application. The start flag icon **320** with title **322** positioned beside it provides familiarity with the start icon from some desktop UIs. As will be discussed later with reference to FIGS. **5–7**, when a start menu is deployed, the title displayed by shell program navigation bar **302** changes from the title of the active application to "Start" in order to reinforce to the user the correlation between the navigation bar and its corresponding functions.

Another significant feature of UI **300** is that the application menu bar **304** is displayed at the bottom of the screen. It has been found that the user's eyes first gravitate to the top of the screen, so it is beneficial to put the "anchoring" data there. Stagnant information such as application commands (available through the menu bar **304**) can be placed in a less vital but still extremely accessible screen location (the bottom). In this manner, when the user wishes to issue a command upon the data in the middle of the screen, he or she

goes to the menu commands at the bottom and his or her hand does not obscure the data. Only when the user is ready to navigate to another application does the user's hand obscure the screen.

FIGS. 5–7 illustrate features of navigation bar **302** provided by UI **300** of the present invention. When the user wishes to implement a shell program function or to switch between application programs, stylus **86** is used to tap navigation icon **320** to deploy navigation menu **324** as is known in the art. However, upon deployment of navigation menu **324**, the shell program changes title **322** from the name of the active application program (see FIG. **4**) to "Start" or to a title which clearly indicates to the user that a shell navigation menu is displayed. This is shown for example in FIG. **5**. Then, if a different application program is activated, the shell program again changes title **322** displayed in navigation bar **302**. For example, in one embodiment, if the "Tasks" item in menu **324** is selected using the stylus as shown in FIG. **6**, upon activation of the Tasks program the title **322** in navigation bar **302** is changed accordingly as shown in FIG. **7**.

Another feature of UI **300** shown in FIG. **7** is the dynamically controlled inclusion of the confirmation ("OK") button **330** at the top right hand corner in navigation bar **302**. Taking advantage of the corners of the screen as places that stand out to the user, in response to a query from the application or shell programs, selecting confirmation button **330** with the stylus dismisses the current screen contents and returns the user to the "owner" (e.g., tapping the confirmation button **330** in an open contact card entry will return the displayed information to the main list). In some embodiments, while confirmation button **330** is displayed in the upper right hand corner of the screen, the clock shifts to the left. Once the confirmation button is no longer displayed, the clock is once again right justified.

FIG. **8** illustrates yet other aspects of UI **300** in accordance with the present invention. As shown in FIG. **8**, menu bar **304** includes menu deploying names or buttons **340** as is known in the art. Tapping one of the menu deploying buttons **340** in menu bar **304** results in deployment of the appropriate menu **345** as is also known in the art. Generally, the user can select desired tools or functions within deployed tool bar menus **345**.

In addition to menu bar **304** being positioned at the bottom of the screen, menu bar **304** in UI **300** can include tool bar icons **350**. Tapping tool icons **350** results in the implementation or activation of functions (i.e, cut, copy, paste, etc.) as is known in the art. Inclusion of both menu deploying buttons **340** and tool implementing icons **350** on a single menu bar **304** saves vertical real estate on the screen, which is highly beneficial in hand-held devices.

To summarize some of the above discussed aspects of the present invention, the present invention includes a UI, for a stylus based hand-held device, which is optimized for navigation and application control while better utilizing screen space. According to a first aspect of the invention, an application specific title is displayed on the shell program's navigation bar. In the prior art, the application specific title has been displayed by the application in a row separate from the shell program's navigation bar, wasting vertical screen real estate. Further, the navigation bar is placed at the top of the screen since when changing between programs, it is unimportant whether the user's hand blocks the displayed data.

According to a second aspect of the present invention, the UI displays toolbar icons **350** in the same row (i.e., menu bar) as menu buttons **340**, thus saving real estate on the display. In some embodiments, this row, which includes the application specific menu buttons and the toolbar icons, is placed at the bottom of the screen so that the user's hand does not block the displayed data when selecting a menu button for deploying a menu or a tool icon for implementing a tool or function which manipulates the data.

According to a third aspect of the present invention, the shell-controlled soft input panel (SIP) button **90** is displayed on the application controlled menu bar **304** at the bottom of the display instead of on the shell program's navigation bar **302** as is conventionally the case.

Although the present invention has been described with reference to particular embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

What is claimed is:

**1**. A method of generating a user interface for a stylus-based hand-held computing device having a display screen and a stylus for providing user input through the screen, the hand-held device running a shell program and at least one application program, the method comprising:

displaying information for an active application program in a middle portion of the screen; and

displaying a shell program controlled navigation bar at a top portion of the screen, wherein displaying the shell program controlled navigation bar includes displaying a navigation icon which when tapped by the stylus aids the user in navigating to other application programs, and wherein displaying the shell program controlled navigation bar further includes displaying on the navigation bar a title for the active program.

**2**. The method of claim **1**, and further comprising:

receiving a first input from the user in the form of a tap on the navigation icon;

displaying a navigation menu in response to the first input, and

displaying on the shell program controlled navigation bar a shell program indicative title in place of the title for the active application program while the navigation menu is displayed.

**3**. The method of claim **2**, wherein displaying the shell program indicative title further includes displaying Start on the shell program controlled navigation bar in place of the title for the active application program while the navigation menu is displayed.

**4**. The method of claim **2**, and further comprising:

receiving a second input from the user selecting an alternate application program as the active program; and

displaying on the shell program controlled navigation bar a title for the alternate application program selected as the active program.

**5**. The method of claim **4**, wherein displaying the title for the alternate application program selected as the active program further includes displaying the title for the alternate program on the navigation bar when the navigation menu is no longer displayed.

**6**. The method of claim **1**, and further comprising displaying a confirmation icon in a corner of the navigation bar, wherein the user provides a response to a query from the active application program by tapping the confirmation icon.

**7**. The method of claim **1**, and further comprising displaying an application menu bar for the active application program at a bottom portion of the screen.

8. The method of claim 7, wherein displaying the application menu bar for the active application program at the bottom portion of the screen further comprises:

displaying menu deploying buttons on the application menu bar, wherein a tap on the menu deploying buttons with the stylus results in an application program menu being displayed; and

displaying a soft input panel icon on the application menu bar, wherein a tap on the soft input panel icon with the stylus controls display of a shell program soft input panel.

9. The method of claim 8, wherein displaying the application menu bar for the active application program further includes displaying tool icons on the application menu bar with the menu deploying buttons and the soft input panel icon.

10. A computer readable medium for a stylus-based hand-held computing device having a display screen and a stylus for providing user input through the screen, the hand-held computing device adapted to run a shell program and at least one application program, the computer readable medium having computer executable instructions for performing steps comprising:

displaying information for an active application program in a middle portion of the screen; and

displaying a shell program controlled navigation bar at a top portion of the screen, wherein displaying the shell program controlled navigation bar includes displaying a navigation icon which when tapped by the stylus aids the user in navigating to other application programs, and wherein displaying the shell program controlled navigation bar further includes displaying on the navigation bar a title for the active program.

11. The computer readable medium of claim 10, and further including computer executable instructions for performing steps comprising:

receiving a first input from the user in the form of a tap on the navigation icon;

displaying a navigation menu in response to the first input; and

displaying on the shell program controlled navigation bar a shell program indicative title in place of the title for the active application program while the navigation menu is displayed.

12. The computer readable medium of claim 11, and further including computer executable instructions for performing steps comprising:

receiving a second input from the user selecting an alternate application program as the active program; and

displaying on the shell program controlled navigation bar a title for the alternate application program selected as the active program.

13. The computer readable medium of claim 12, wherein the computer executable instructions for displaying the title for the alternate application program selected as the active program further includes computer executable instructions for displaying the title for the alternate program on the navigation bar when the navigation menu is no longer displayed.

14. The computer readable medium of claim 10, and further including computer executable instructions for displaying a confirmation icon in a corner of the navigation bar, wherein the user provides a response to a query from the active application program by tapping with the stylus on the confirmation icon.

15. The computer readable medium of claim 10, and further including computer executable instructions for displaying an application menu bar for the active application program at a bottom portion of the screen.

16. The computer readable medium of claim 15, wherein the computer executable instructions for displaying the application menu bar for the active application program at the bottom portion of the screen further includes computer executable instructions for performing steps comprising:

displaying menu deploying buttons on the application menu bar, wherein a tap on the menu deploying buttons with the stylus results in an application program menu being displayed; and

displaying a soft input panel icon on the application menu bar, wherein a tap on the soft input panel icon with the stylus controls display of a shell program soft input panel.

17. The computer readable medium of claim 16, wherein the computer executable instructions for displaying the application menu bar for the active application program further includes computer executable instructions for displaying tool icons on the application menu bar with the menu deploying buttons and the soft input panel icon.

18. A method of generating a user interface for a stylus-based hand-held computing device having a display screen and a stylus for providing user input through the screen, the hand-held device running a shell program and at least one application program, the method comprising:

displaying information for an active application program in a middle portion of the screen; and

displaying an application menu bar for the active application program at a bottom portion of the screen such that the user can manipulate information for the active application program displayed in the middle portion of the screen by tapping with the stylus on the application menu bar without blocking the user's view of the middle portion of the screen, wherein displaying the application menu bar for the active application program further comprises displaying a soft input panel icon on the application menu bar, wherein a tap on the soft input panel icon with the stylus controls display of a shell program soft input panel.

19. The method of claim 18, wherein displaying the application menu bar further comprises: displaying menu deploying buttons on the application menu bar, wherein a tap on the menu deploying buttons with the stylus results in an application program menu being displayed.

20. The method of claim 19, wherein displaying the application menu bar for the active application program further includes displaying tool icons on the application menu bar with the menu deploying buttons and the soft input panel icon.

* * * * *

US006738773B1

(12) **United States Patent**
**Reunert et al.**

(10) **Patent No.: US 6,738,773 B1**
(45) **Date of Patent: May 18, 2004**

(54) **METHOD AND SYSTEM FOR TRANSFORMING DATA BETWEEN DISPARATE CAPACITY DATABASE SYSTEMS**

(75) Inventors: **Philip P. Reunert**, Redmond, WA (US); **Mohanraj Dharmarajan**, Redmond, WA (US); **Jeffrey G. Ort**, Redmond, WA (US); **Peter Zatloukal**, Duvall, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

( * ) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/562,992**

(22) Filed: **May 1, 2000**

(51) **Int. Cl.**[7] ..... **G06F 17/30**; G06F 7/00
(52) **U.S. Cl.** ..... **707/10**; 707/8; 707/1; 707/104.1; 707/200; 707/201
(58) **Field of Search** ..... 707/1–10, 100–104.1, 707/200–205; 709/200–225, 210–231

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 5,504,899 | A | * 4/1996 | Raz | 707/10 |
| 5,546,579 | A | * 8/1996 | Josten et al. | 700/5 |
| 5,806,076 | A | * 9/1998 | Ngai et al. | 707/203 |
| 5,819,083 | A | * 10/1998 | Chen et al. | 370/235 |
| 5,850,507 | A | * 12/1998 | Ngai et al. | 707/202 |
| 5,862,327 | A | * 1/1999 | Kwang et al. | 707/10 |
| 5,930,794 | A | * 7/1999 | Linenbach et al. | 707/100 |
| 5,944,796 | A | * 8/1999 | Noritomi | |
| 5,961,606 | A | * 10/1999 | Talluri et al. | |
| 5,978,577 | A | * 11/1999 | Rierden et al. | 340/825.52 |
| 6,003,043 | A | * 12/1999 | Hatakeyama et al. | 707/102 |
| 6,058,413 | A | * 5/2000 | Flores et al. | 705/7 |
| 6,131,115 | A | * 10/2000 | Anderson et al. | 705/40 |
| 6,157,932 | A | * 12/2000 | Klein et al. | 707/200 |
| 6,202,067 | B1 | * 3/2001 | Blood et al. | 707/10 |
| 6,240,417 | B1 | * 5/2001 | Eastwick et al. | 707/1 |

* cited by examiner

*Primary Examiner*—Srirama Channavajjala
(74) *Attorney, Agent, or Firm*—Woodcock Washburn LLP

(57) **ABSTRACT**

In many computer systems, the rate of data from an input source may become significantly faster than a processor's ability to respond. This situation occurs when a faster and more modern system writes to an older and slower legacy system. The result of this interaction between disparate capacity systems may be system failure or data corruption, particularly due to overloading the slower system's input queue to the point where it can no longer catch up to its workload. The invention provides a method, system, and computer-readable medium having computer-executable instructions for transferring data between disparate capacity systems. The method comprises the steps of receiving transactions from a first computer system at a first rate and providing the transactions to a second computer system, wherein the second computer system receives the transactions at a second rate. The second computer system may be a database legacy database server and the first computer system may be an updated database server The second rate is set at a capacity of the second computer system. The method further delays for a predetermined period of time the transactions when the first rate exceeds the second rate. The predetermined period of time is a function of the second rate.

**27 Claims, 6 Drawing Sheets**

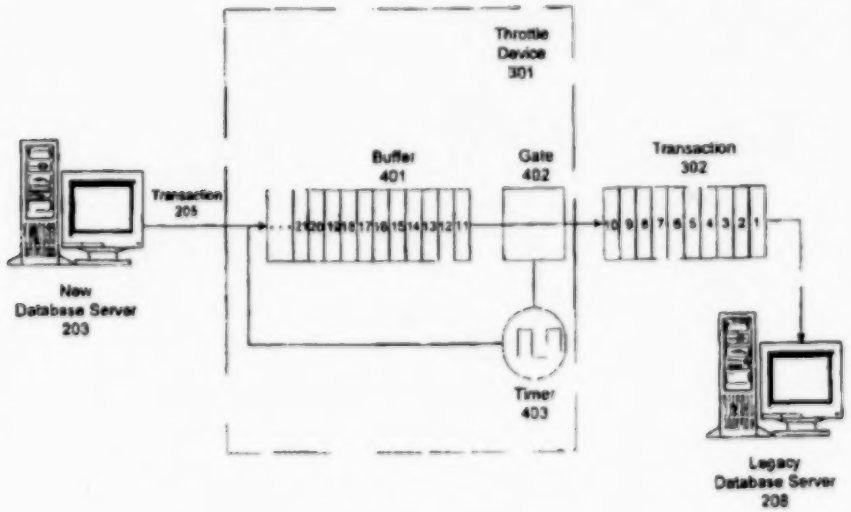
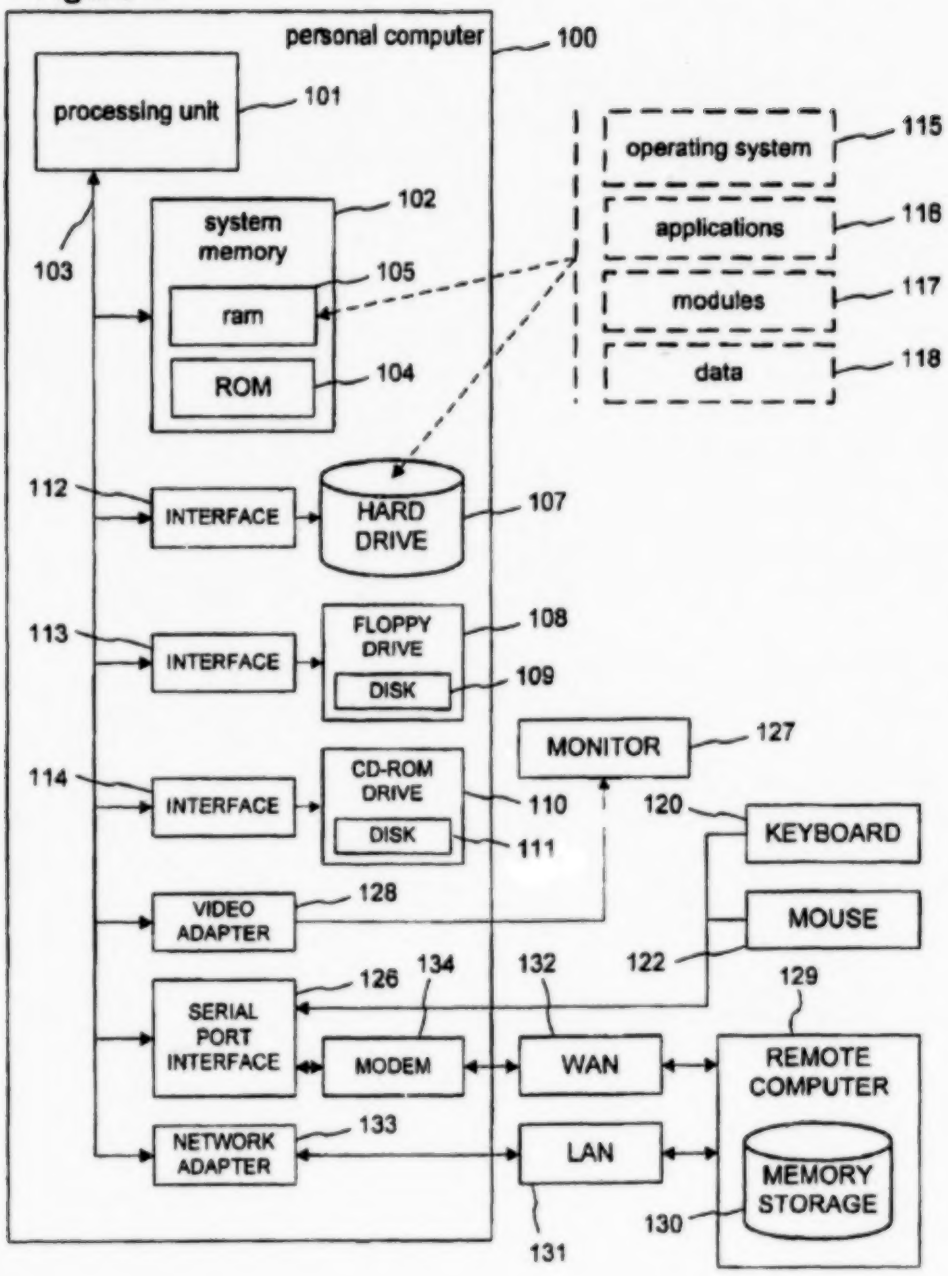


Figure 1


personal computer
100
processing unit
101
system memory
102
ram
105
ROM
104
103
operating system
115
applications
116
modules
117
data
118
112
INTERFACE
HARD DRIVE
107
113
INTERFACE
FLOPPY DRIVE
108
DISK
109
114
INTERFACE
CD-ROM DRIVE
110
DISK
111
MONITOR
127
120
KEYBOARD
MOUSE
122
VIDEO ADAPTER
128
SERIAL PORT INTERFACE
126
134
MODEM
132
WAN
129
REMOTE COMPUTER
NETWORK ADAPTER
133
LAN
MEMORY STORAGE
130
131

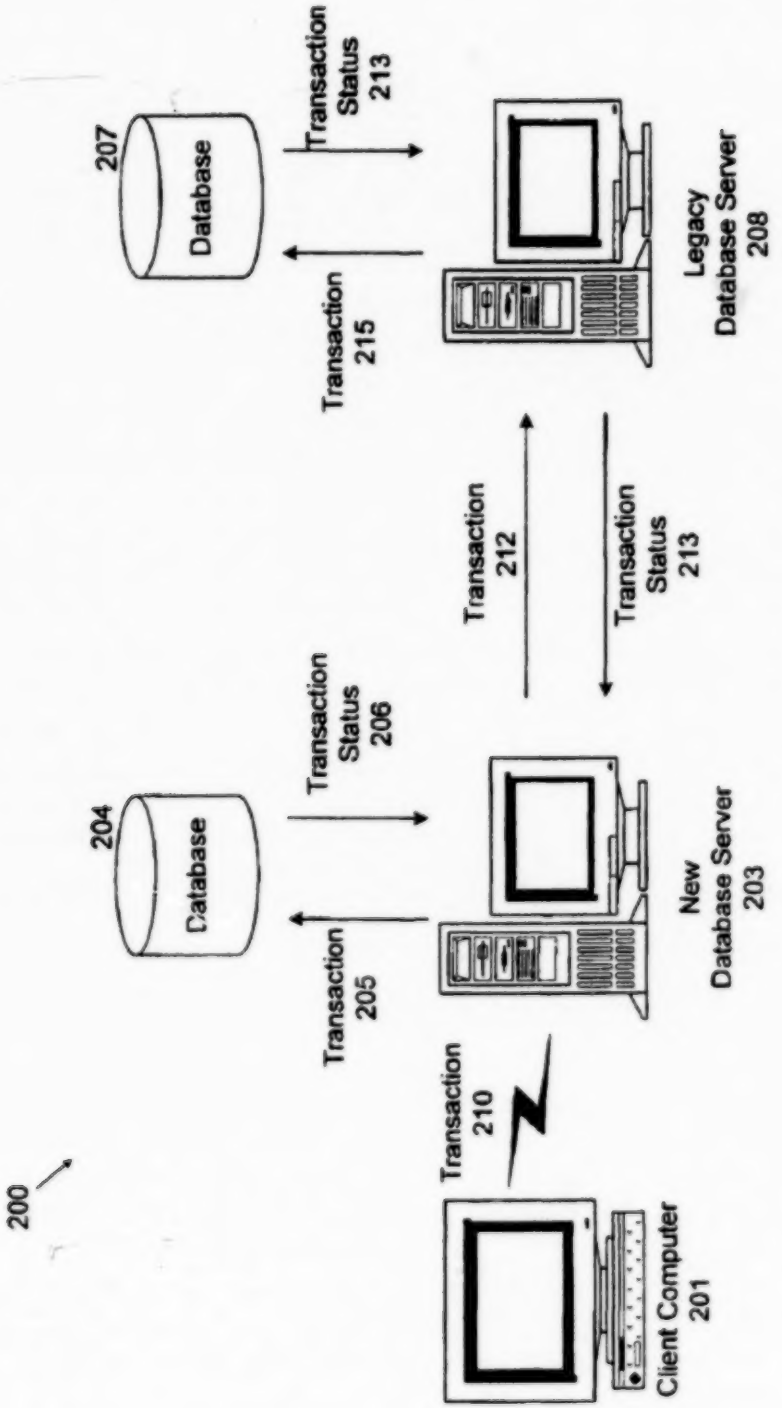

200
Client Computer
201
Transaction
210
Transaction
205
New
Database Server
203
Transaction
Status
206
204
Database
Transaction
212
Transaction
Status
213
Transaction
215
Legacy
Database Server
208
Transaction
Status
213
207
Database

Figure 2

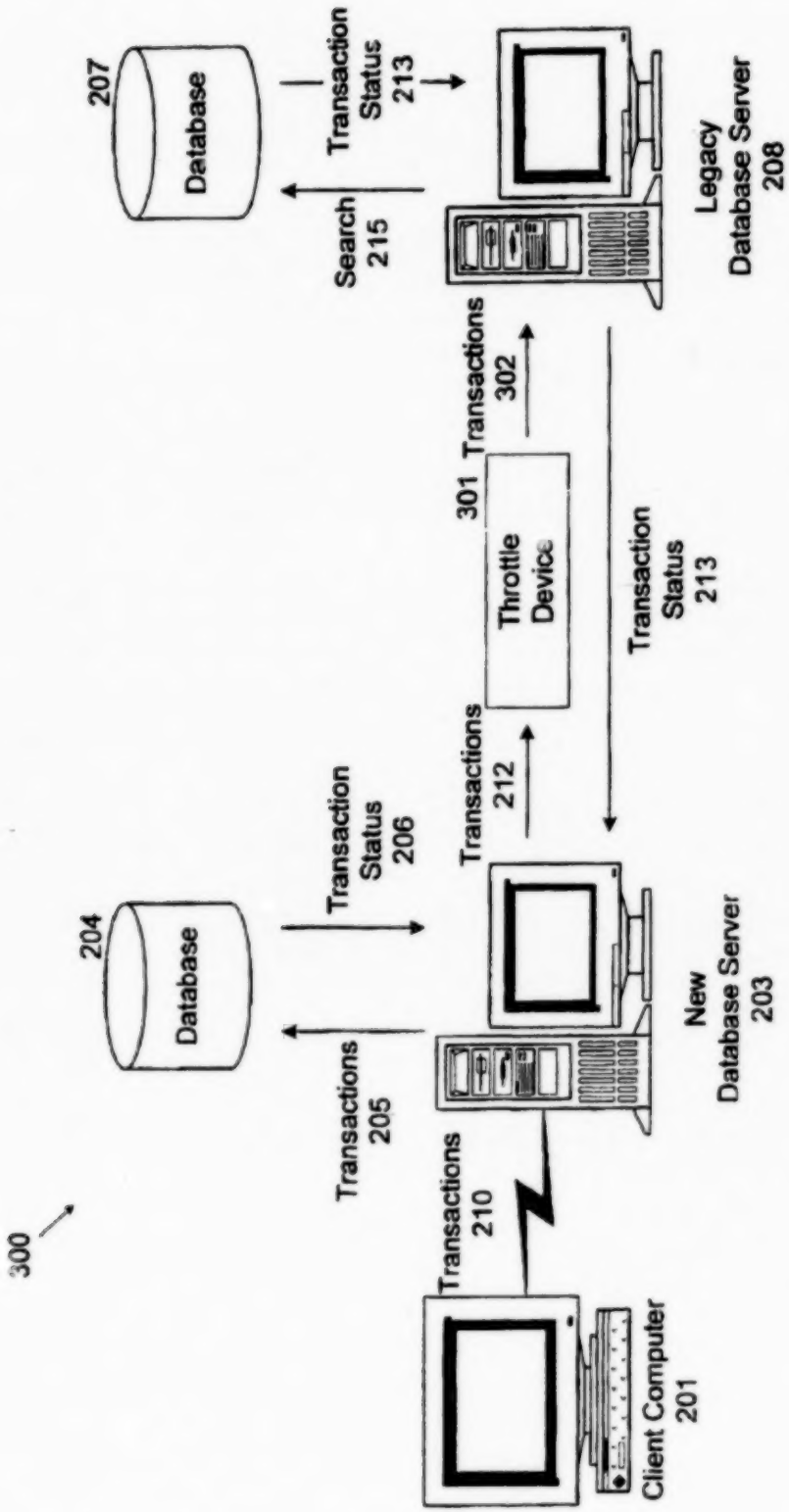

300
Client Computer
201
Transactions
210
Transactions
205
Database
204
Transaction
Status
206
New
Database Server
203
Transactions
212
Throttle
Device
301
Transactions
302
Transaction
Status
213
Transaction
Status
213
Search
215
Database
207
Legacy
Database Server
208

Figure 3

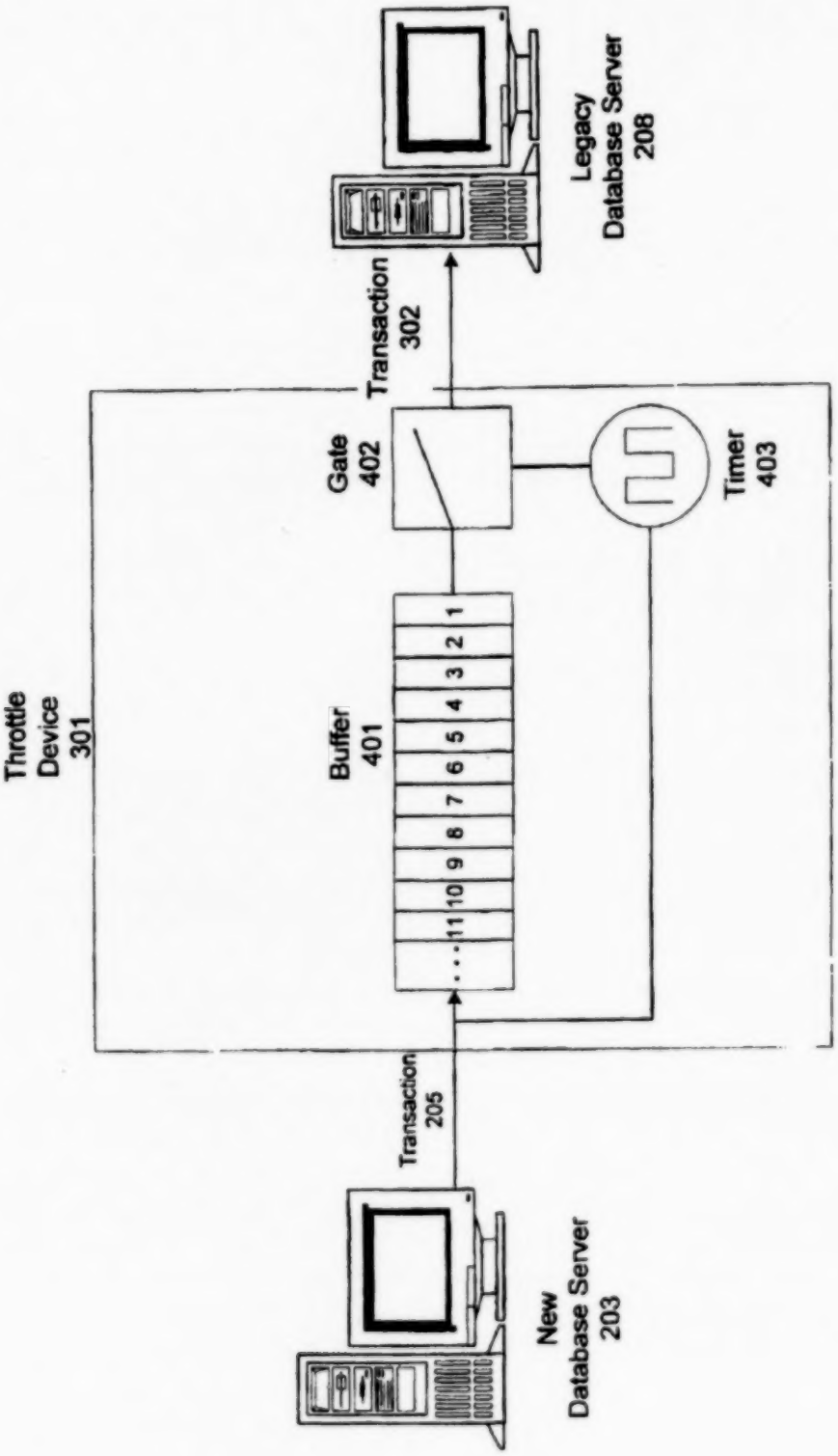

Legacy
Database Server
208
Transaction
302
Gate
402
Timer
403
Throttle
Device
301
Buffer
401
1
2
3
4
5
6
7
8
9
10
11
Transaction
205
New
Database Server
203

Figure 4

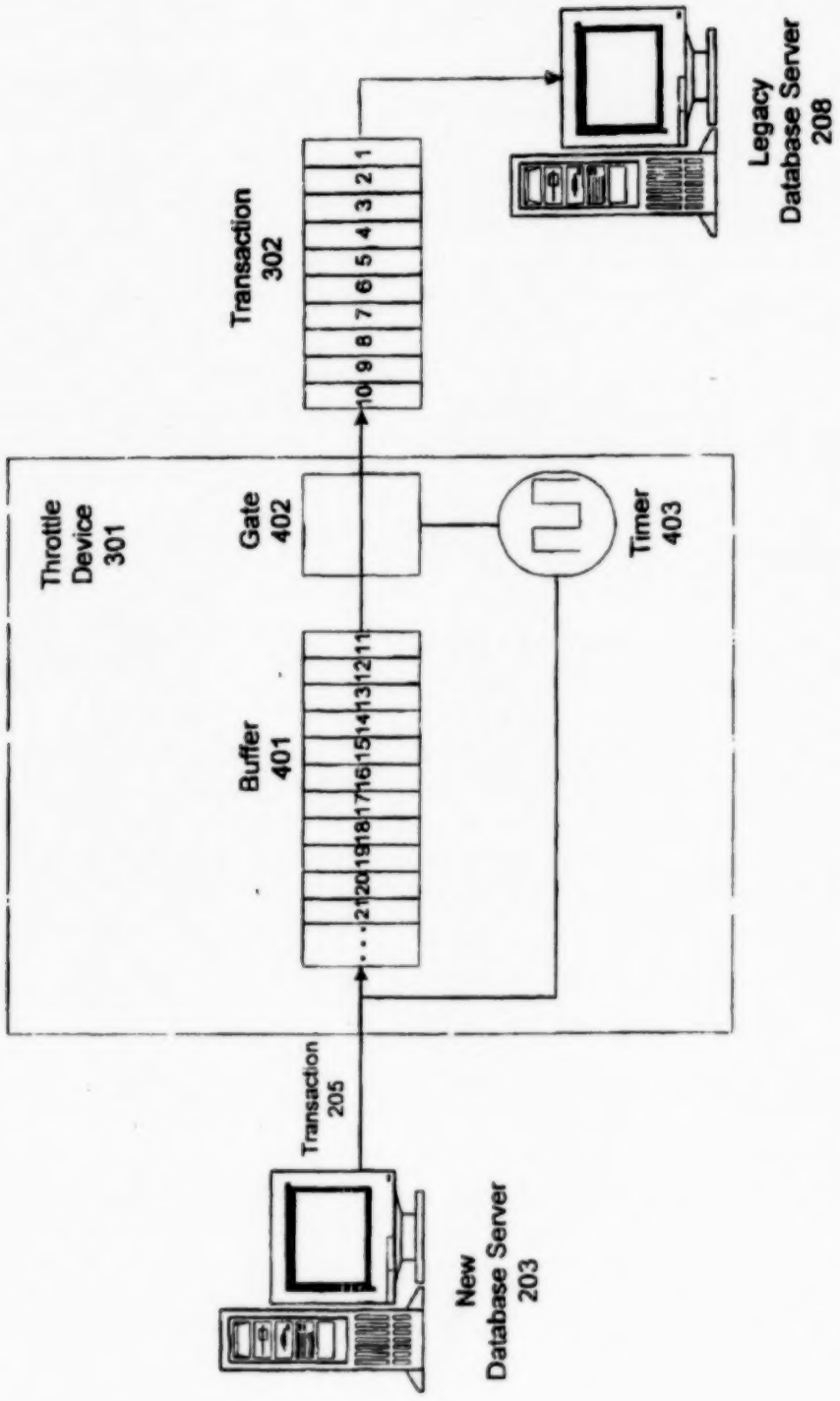

Legacy Database Server 208
Transaction 302
1 2 3 4 5 6 7 8 9 10
Throttle Device 301
Gate 402
Timer 403
Buffer 401
11 12 13 14 15 16 17 18 19 20 21
Transaction 205
New Database Server 203

Figure 5

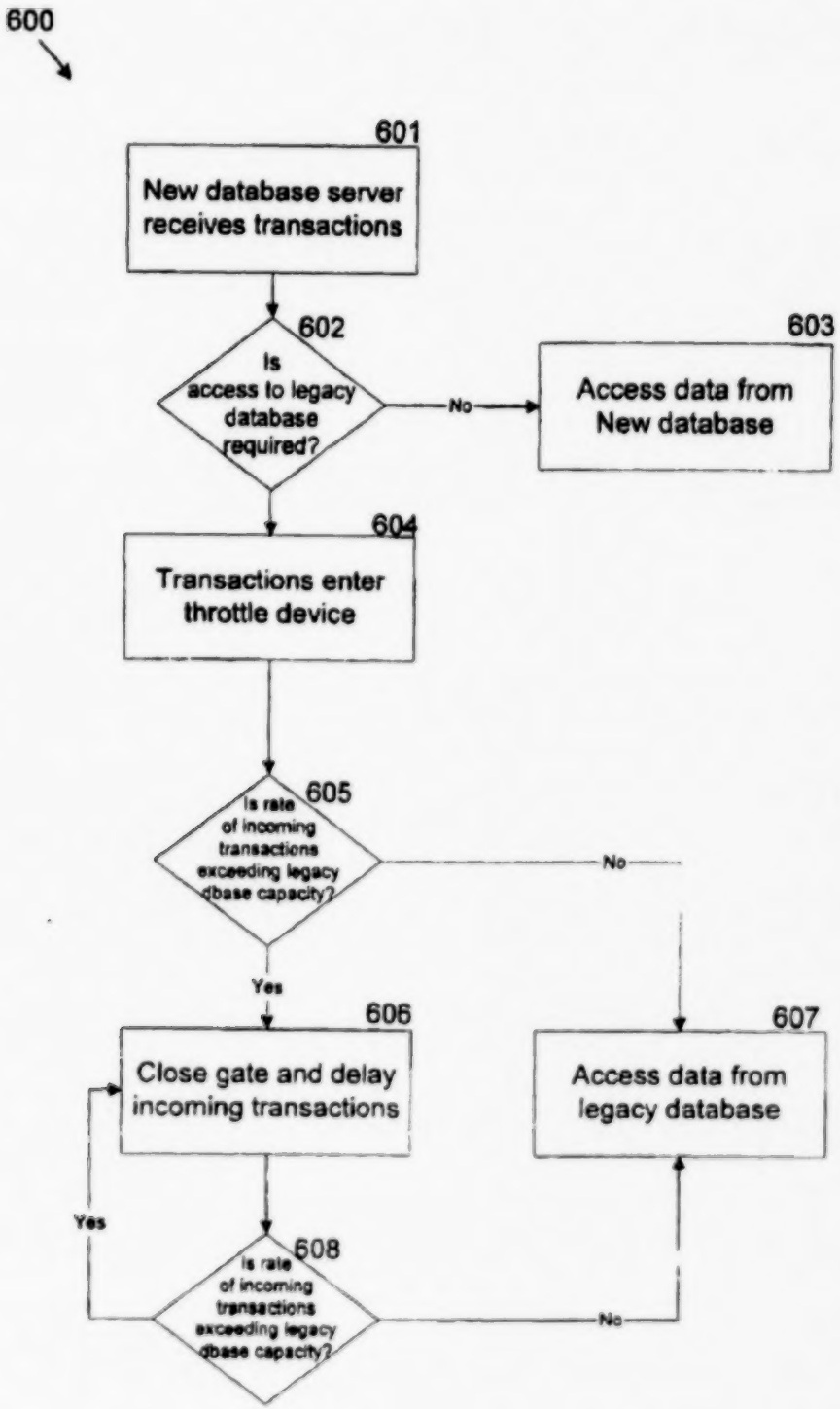

600
601
New database server receives transactions
602
Is access to legacy database required?
No
603
Access data from New database
604
Transactions enter throttle device
605
Is rate of incoming transactions exceeding legacy dbase capacity?
No
Yes
606
Close gate and delay incoming transactions
607
Access data from legacy database
Yes
608
Is rate of incoming transactions exceeding legacy dbase capacity?
No

Figure 6

# METHOD AND SYSTEM FOR TRANSFORMING DATA BETWEEN DISPARATE CAPACITY DATABASE SYSTEMS

## TECHNICAL FIELD

The present invention relates to the field of data transfer. More specifically, the present invention relates to transferring data between disparate capacity database systems.

## BACKGROUND OF THE INVENTION

A database system is a collection of information organized in such a way that data may be accessed quickly. Database applications are present in everyday use including automated teller machines, flight reservation systems, and internet-account systems of all types. A database management system (DBMS) is a collection of programs that allow the entry, modification, and selection of data in a database. There are various types of DBMSs, ranging from small systems that run on personal computers to larger systems that run on mainframe systems.

Requests for information from a database are made by a database server using a query language. Different DBMSs support different query languages, including a standardized language called "structured query language" (SQL), for example. Database servers that use SQL are often referred to as "SQL servers." SQL supports distributed database systems, which are databases that are spread out over several server systems, yet act as one. A distributed database system must replicate and synchronize data among its many databases. In this way, a distributed database system may enable several users on a network to access the same database simultaneously without interfering with one another. Therefore, a distributed database system may have many SQL servers located in various locations, each responsible for certain portions of the database's overall operation.

In distributed database systems often there occurs a disparity in the relative capacity to perform work among the individual databases in the system. This can happen for a variety of reasons, including but not limited to new demand conditions, addition of newer hardware in one subsystem which is substantially faster than the legacy systems, and progressive redesign and redeployment of portions of an existing system. Moreover, it may be more efficient to gain the functionality of a new distributed database by dividing the database's operation between a newer and faster database server and the legacy server. In either case, the result is that various versions of database servers simultaneously may exist in one database system.

Although having multiple versions of SQL servers in one database system may be more cost efficient, often this scenario causes technical concerns. This is particularly true, for example, where a database system is designed to allow a faster upgraded database server to automatically transact with a slower legacy database that may be in various states of repair. In this instance, transactions may be received from the upgraded server at a rate faster than the legacy server can process them. This disparate capacity is particularly troubling in a distributed database system where the overwhelming rate of transactions entering the legacy system may break down the replication process and return corrupted or stale data to the upgraded server.

To date, when confronted with these disparate server systems, administrators are forced to upgrade the slower legacy system to be compatible with the faster upgraded system. Therefore, there exists a need to regulate incoming transactions from the upgraded server to a rate that is compatible with the slower legacy server.



## SUMMARY OF THE INVENTION

In many computer systems, the rate of data from an input source may become significantly faster than a processor's ability to respond. This situation occurs, for example, when a faster and more modern system writes to an older and slower legacy system. Such interactions between disparate capacity systems may result in failure or data corruption, particularly due to overloading the slower system's input queue to the point where it can no longer catch up to its workload. For example, in a database computer system, when the rate of incoming transactions from an updated database system exceeds a smaller capacity database system's ability to process transactions, system failure may occur. In some instances when this situation arises, system administrators may upgrade the legacy database system to operate with the updated database system. In running production systems, this is often operationally unfeasible, or not feasible in a reasonable time frame. Where system upgrade is impossible or unfeasible, there exists a need to slow incoming transactions down to a rate compatible with the slower legacy system.

The invention provides a method, system, and computer-readable medium having computer-executable instructions for transferring data between disparate capacity systems. The method comprises the steps of receiving transactions from a first computer system at a first rate and providing the transactions to a second computer system, wherein the second computer system receives the transactions at a second rate. The second computer system may be a database legacy database server and the first computer system may be an updated database server The second rate is set at a capacity of the second computer system. The method further delays for a predetermined period of time the transactions when the first rate exceeds the second rate. The predetermined period of time is a function of the second rate.

## BRIEF DESCRIPTION OF THE DRAWINGS

Other features of the invention are further apparent from the following detailed description of presently preferred embodiments of the invention taken in conjunction with the accompanying drawings, of which:

FIG. **1** is a block diagram of a suitable computing environment in which the present invention may be implemented;

FIG. **2** is a block diagram of a client-server database system in which the present invention may be implemented;

FIG. **3** is a block diagram of a client-server database system, according to the present invention;

FIG. **4** provides a schematic diagram of a throttle device, according to the present invention;

FIG. **5** provides another schematic diagram of a throttle device, according to the present invention; and

FIG. **6** provides a flow diagram of a method for transferring data between disparate capacity databases in a client-server database system, according to the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

### Operating Environment

FIG. **1** and the following discussion are intended to provide a brief, general description of a suitable computing

environment in which the invention may be implemented. While the invention will be described in the general context of computer-executable instructions of a computer program that runs on a personal computer, those skilled in the art will recognize that the invention also may be implemented in combination with other program modules. Generally, program modules include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The illustrated embodiment of the invention also is practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. However, some embodiments of the invention can be practiced on standalone computers. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to FIG. **1**, one system for implementing the invention includes a conventional personal computer **100**, including a processing unit **101**, a system memory **102**, and a system bus **103** that couples various system components including the system memory to the processing unit **101**. Processing unit **101** may be any of various commercially available processors. Dual microprocessors and other multiprocessor architectures also can be used as processing unit **101**.

System bus **103** may be any of several types of bus structure including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of conventional bus architectures. System memory **102** includes read only memory (ROM) **104** and random access memory (RAM) **105**. A basic input/output system (BIOS), containing the basic routines that help to transfer information between elements within the personal computer **100**, such as during start-up, is stored in ROM **104**.

Personal computer **100** further includes a hard disk drive **107** and a magnetic disk drive **108** to read from or write to a removable disk **109**, and an optical disk drive **110** to read a CD-ROM disk **111** or to read from or write to other optical media. Hard disk drive **107**, magnetic disk drive **108**, and optical disk drive **110** are connected to system bus **103** by a hard disk drive interface **112**, a magnetic disk drive interface **113**, and an optical drive interface **114**, respectively. The drives and their associated computer-readable media provide nonvolatile storage of data, data structures, computer-executable instructions, etc. for personal computer **100**. Although the description of computer-readable media above refers to a hard disk, a removable magnetic disk and a CD, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the like, may also be used in the operating environment.

A number of program modules may be stored in the drives and RAM **105**, including an operating system **115**, one or more application programs **116**, other program modules **117**, and program data **118**.

A user may enter commands and information into personal computer **100** through a keyboard **120** and pointing device, such as a mouse **122**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to processing unit **101** through a serial port interface **126** that is coupled to system bus **103**, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor **127** or other type of display device is also connected to system bus **103** via an interface, such as a video adapter **128**. In addition to monitor **127**, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

Personal computer **100** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **129**. Remote computer **129** may be a server, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to personal computer **100**, although only a memory storage device **130** has been illustrated in FIG. **2**. The logical connections depicted in FIG. **2** include a local area network (LAN) **131** and a wide area network (WAN) **132**. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, personal computer **100** is connected to local network **131** through a network interface or adapter **133**. When used in a WAN networking environment, personal computer **100** typically includes a modem **134** or other means for establishing communications over wide area network **132**, such as the Internet. Modem **134**, which may be internal or external, is connected to system bus **103** via serial port interface **126**. In a networked environment, program modules depicted relative to personal computer **100**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are one example, and that other means of establishing a communications link between the computers may be used.

In accordance with the practices of persons skilled in the art of computer programming, the present invention is described below with reference to acts and symbolic representations of operations that are performed by personal computer **100**, unless indicated otherwise. Such acts and operations are sometimes referred to as being computer-executed. It will be appreciated that the acts and symbolically represented operations include the manipulation by the processing unit **101** of electrical signals representing data bits which causes a resulting transformation or reduction of the electrical signal representation, and the maintenance of data bits at memory locations in the memory system (including system memory **102**, hard drive **107**, floppy disks **109**, and CD-ROM **111**) to thereby reconfigure or otherwise alter the computer system's operation, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, or optical properties corresponding to the data bits.

Database Client-Server System

FIG. **2** is a block diagram of a client-server database system **200** in which the present invention may be implemented. Client-server system **200** includes a client computer **201** coupled to a new database server **203**. Client computer **201** may be a personal computer (as shown in FIG. **1**) containing software that allows a user to access a database, for example, ACCESS available from MICROSOFT Corporation. It should be appreciated that while one client computer **201** is shown in FIG. **2**, in practice, there may be many client computers simultaneously accessing new data-

base processor **203**. Client computer **201** sends a transaction **210** to new database server **203**. The connection between client computer **201** and new database processor **203** may be a LAN or WAN, for example, the Internet. New database server **203** is coupled to an older, slower legacy database server **208**. New database server **203** may be an upgraded and updated version of legacy database server **208**, both in terms of faster hardware and/or next generation software. New database server **203** is coupled to database **204**, and legacy database server **208** is coupled to database **207**.

In operation, client computer **201** sends a transaction **210** to new database server **203**. It should be appreciated that transaction **210** may include a set of instructions necessary to execute a particular operation. The set of instructions may include both a request to read data from a database and a request to write data to a database. Once new database server **203** receives transaction **210** from computer client **201**, new database server **203** determines whether completion of transaction **210** requires access to data located in database **204** or database **207**. If transaction **210** requires access to data located in database **204**, new database server **203** sends a transaction **205** to database **204**. Upon completion of transaction **205**, new database server **203** receives a transaction status **206** indicating that transaction **205** is complete. Alternatively, transaction status **206** may provide an error signal to new database server **203** indicating that transaction **205** was not completed. If new database server **203** determines that some or all of transaction **210** requires access to data located in database **207**, new database server **203** sends a transaction **212** to legacy database server **208**. Legacy database server **208** then sends a transaction **215** to database **207**. Legacy database server **208** receives a transaction status **213** indicating that transaction **212** is complete. Alternatively, transaction status **213** may provide an error signal to new database server **203** indicating that transaction **212** was not completed.

Data Throttling Operation

FIG. **3** is a block diagram of a database client-server system **300**, according to the present invention. FIG. **3** is similar to the block diagram shown in FIG. **2**, except that a throttle device **301** is coupled between new database server **203** and legacy database server **208**. Although throttle device **301** is shown separate from new database server **203**, it should be appreciated that throttle device **301** may be a hardware or software component of new database server **203** or of legacy database server **208**. As shown in FIG. **3**, transactions **205** from new database server **203** enter throttle device **301**. Transactions **302** are then provided by throttle device **301** to legacy database server **208**. Because new database server **203** is an upgraded and updated version of older legacy database server **208**, the rate that at which transactions **205** are received from new database server **203** may be faster than the rate at which legacy database server **208** can process incoming transactions **302**. Therefore, throttle device **301** regulates received transactions **205** to a rate that is compatible with the processing speed of slower legacy database server **208**.

FIG. **4** provides a schematic diagram of throttle device **301**, according to the invention. As discussed, new database server **203** is capable of providing transactions **205** at a rate faster than the rate at which legacy database server **208** is capable of processing incoming transactions **302**. Because the capacity of legacy database server **208** is known to the database system administrator, throttle device **301** can be set to monitor the rate of incoming transactions **205** from new database server **203**, and delay incoming transactions **205** when they arrive at a rate that exceeds legacy database server's **208** predetermined capacity. Alternatively, throttle device **301** will allow transactions **205** to pass to legacy database server **208** without delay, when incoming transactions **205** arrive at a rate less than legacy database server's **208** predetermined capacity.

As shown in FIG. **4**, throttle device **301** includes a buffer **401** (e.g., a first-in first-out queue) that is coupled to a gate **402**. A timer **403** is coupled to gate **402** and to incoming transactions **205**. Timer **403** is set to permit transactions **205** to enter legacy database server **208** at a rate consistent with the processing speed of legacy database server **208**. When transactions **205** begin to exceed the predetermined rate at which legacy database server **208** is capable of processing transactions **302**, timer **403** closes gate **402**. As shown in FIG. **4**, when gate **402** is closed (i.e., not allowing transactions in buffer **401** to pass onto legacy database server **208**), transactions **205** provided by new database server **203** enter throttle device **301** and are stored in buffer **401**. In FIG. **4**, each of buffered transactions **205** are numbered (e.g., 1 through 11) in the order in which they are received. The number of actual transactions **205** that buffer **401** can store will be determined by the rate at which transactions **205** are provided by new database server **203** and the rate at which transactions **302** can be processed by slower legacy database server **208**. For example, FIG. **4** shows at least eleven received queries **205** located in buffer **401**. In effect, therefore, timer **403** allows transactions queued in buffer **401** to pass to legacy database server **208** at the same rate, regardless of the rate of incoming transactions **205**. If the rate of incoming transactions **205** is less than the processing rate of legacy database server **208**, transactions **205** will pass through throttling device **301** without delay. If, on the other hand, the rate of incoming transactions **205** is greater than the processing rate of legacy database server **208**, throttling device **301** will begin to store transactions **205** in buffer **401**.

Notably, the predetermined rate at which legacy database server **208** can process transactions **302** may be set, and dynamically adjusted, in timer **403**. For example, if legacy database server **208** were replaced with a system having more or less processing capacity, an administrator could adjust the predetermine rate in timer **403** to send the transactions more or less often, as the case may be.

FIG. **5** provides a detailed schematic diagram of throttle device **301** in which gate **402** is shown open . If timer **403** notices that transactions **205** are not exceeding the rate at which legacy database server can process, timer **403** will open gate **402**, thus allowing transactions **302** to enter legacy database server **208**. As shown in FIG. **5**, transactions 1 through 10 have been sent to legacy database server **208** by throttle device **301**.

The following code provides an example of how throttle device **301** may operate:

The timer is represented by the following code fragment:

```
HRESULT WatchDogThreadFunction(LPVOID lpParam)
{
  HRESULT hResult = S_OK;
  DWORD   dwWatchDogWaitTime = 0;
  DWORD   dwWaitResult = 0;
  do
  {
    dwWaitResult = WaitForSingleObject( g_h WatchDogEvent,
g_ulThrottleDelayInterval);
  if ( WAIT_TIMEOUT == dwWaitResult )
```

-continued

```
        SetEvent( g_hSerializeEvent);
      } while ( FALSE == g_bKillWatchDogThread );
      return hResult;
    }
```

The above timer sends a signal every g_ulThrottleDelayInterval to the main process shown below:

```
    UINT MainQProcessorThreadFunction( LPVOID pParam)
    {
      do
      {
        do
        {
      //wait on serialize event to get signalled
          dwWaitResult = WaitForSingleObject( g_hSerializeEvent,
    SERIALIZE_EVENT_WAIT);
        } while (( WAIT_TIMEOUT == dwWaitResult) &&
    (FALSE == pDriver->GetStopThreadsFlag( )));
        DoWork( );
      } while (FALSE == pDriver->GetStopThreadsFlag( ));
      return 0;
    }
```

The example code has three main components. The first main component is a "throttle thread," respresented by "HRESULT WatchDogThreadFunction(LPVOID lpParam))." The "throttle thread" establishes the timer function of throttle device **301**. The "throttle thread" includes a "throttle event" represented by "g_hWatchDogEvent" and a "throttle value" represented by "g_ulThrottleDelayInterval." The second component is a "worker thread," respresented by "UINT MainQProcessorThreadFunction(LPVOID pParam)." The "worker thread" processes queue messages and provides them to legacy database server **208**. The third main component is called a "signalwork" event, and is represented by "g_hSerializeEvent." The "worker thread" waits on "signalwork" indefinitely. When the "throttle event" times out, as determined by the "throttle value," the "throttle event" signals the "signalwork" to allow the "worker thread" to process a next transaction. Increasing the predetermined interval set by the "throttle value" allows the "worker thread" to be executed at a slower rate, commensurate with legacy database server **208**.

FIG. **6** provides a flow diagram **600** of a method for transferring data between disparate capacity databases in a client-server database system. In step **601**, client computer **201** provides transactions **205** to new database server **203**. In step **602**, new database server **203** determines whether any of transactions **205** require access to database **204** or legacy database **207**. If transactions **205** require access to database **204**, new database server **203** accesses database **204**.

If, on the other hand, transactions **205** require access to legacy database **207**, new database server **203** sends transactions **205** to throttle device **301**, in step **604**. In step **605**, it is determined whether the rate of incoming transactions **205** has exceeded legacy database server's **208** capacity. If the rate of incoming transactions has not exceeded legacy database server's **208** capacity, data is accessed from legacy database **207**, in step **607**. If, on the other hand, the rate of incoming transactions has exceeded legacy database server's **208** capacity, in step **606**, gate **402** is opened and incoming transactions **205** are delayed from entering legacy database server **208**.

In step **608**, once it is determined that the rate of incoming transactions **205** has not exceeded legacy database server's **208** capacity, data is accessed from legacy database **207**, in step **607**.

The present invention is directed to a system and method for transferring data between disparate capacity systems, but is not limited to database components, regardless of any specific description in the drawing or examples set forth herein. It will be understood that the present invention is not limited to use of any of the particular components or devices herein. Indeed, this invention can be used in any application that transfers data between disparate capacity systems or components. For example, although the present invention was described in the context of disparate capacity systems, it should be appreciated that the invention may apply to equivalent capacity systems that are run at different speeds to increase overall performance. Furthermore, the system disclosed in the present invention can be used with the method of the present invention or a variety of other applications.

While the present invention has been particularly shown and described with reference to the presently preferred embodiments thereof, it will be understood by those skilled in the art that the invention is not limited to the embodiments specifically disclosed herein. Those skilled in the art will appreciate that various changes and adaptations of the present invention may be made in the form and details of these embodiments without departing from the true spirit and scope of the invention as defined by the following claims.

We claim:

**1**. A method for transferring data between disparate capacity database systems, comprising the steps of:

receiving transactions from a first database system at a first rate;

providing said transactions to a second database system, wherein said second database system is capable of receiving said transactions at a second rate; and

determining if said transactions provided by said first database system at said first rate exceed said second rate of said second database system; and

delaying for a predetermined period of time said transactions when said first rate exceeds said second rate, wherein said predetermined period of time is adjusted as a function of said capacity of said second database system.

**2**. The method of claim **1**, wherein said predetermined period of time is a function of said second rate.

**3**. The method of claim **1**, wherein said second database system is a transaction processor and said first database system is a data retrieval/entry device.

**4**. The method of claim **1**, wherein said second database system is a legacy database server and said first database system is an updated database server.

**5**. The method of claim **1**, wherein said predetermined period of time is a function of a processing speed of said second database system.

**6**. The method of claim **1**, wherein said predetermined period of time is a function of a capability of said second database system to process said transactions.

**7**. A device for interfacing disparate capacity computer systems, comprising:

an input component for receiving transactions from a first database system at a first rate;

an output component for providing transactions to a second database system at a second rate; and

a buffer component coupled to said input component and said output component, wherein said buffer component determines if said transactions provided by said first database system at said first rate exceed said second rate of said second database system, and wherein said buffer component delays said transactions for a prede-

termined period of time when said first rate exceeds said second rate, wherein said second rate is set at a capacity of said second database system.

8. The device of claim 7, wherein said transactions include data requests and/or data entries.

9. The device of claim 7, wherein said predetermined period of time is a function of said second rate.

10. The device of claim 7, wherein said second database system is a legacy database server and said first database system is an updated database server.

11. The device of claim 7, wherein said first database system is a data retrieval/entry device, and wherein said second database system is a transaction processor.

12. The device of claim 7, wherein said predetermined period of time is a function of a processing speed of said second database system.

13. A disparate capacity computing system, comprising:

a first database system that generates transactions at a first rate;

a queue coupled to said first database system, wherein said queue receives said transactions from said first database system;

a gate device coupled to said queue, wherein said gate device permits said transactions to exit said queue at a predetermined rate; and

a second database system coupled to said gate device, wherein said second database system processes said transactions at a second rate.

14. The system of claim 13, wherein said transactions include data requests and/or data entries.

15. The system of claim 13, wherein said predetermined rate is a function of said second database system.

16. The system of claim 13, wherein said second database system is a transaction processor and wherein said first database system is a data retrieval/entry device.

17. The system of claim 13, further comprising inputting said transactions to said second database system after said predetermined delay when said first rate exceeds said second rate.

18. The system of claim 13, wherein said predetermined rate is a function of a processing speed of said second database system.

19. The system of claim 13, wherein said predetermined rate is a function of a capability of said second database to process said transactions.

20. A computer-readable medium having computer-executable instructions for performing the steps of:

receiving transactions from a first database system at a first rate;

providing said transactions to a second database system, wherein said second database system receives said transactions at a second rate;

determining if said transactions provided by said first database system at said first rate exceed said second rate of said second database system; and

delaying for a predetermined period of time said transactions when said first rate exceeds said second rate, wherein said predetermined period of time is adjusted as a function of said capacity of said second database system.

21. The computer-readable medium of claim 20, wherein said predetermined period of time is a function of said second rate.

22. The computer-readable medium of claim 20, wherein said second rate is set at a capacity of said second database system.

23. The computer-readable medium of claim 20, wherein said second database system is a transaction processor and said first database system is a data retrieval/entry device.

24. The computer-readable medium of claim 20, wherein said second database system is a legacy database server and said first database system is an updated database server.

25. The computer-readable medium of claim 20, wherein said predetermined period of time is a function of a processing speed of said second database system.

26. The computer-readable medium of claim 20, wherein said predetermined period of time is a function of a capability of said second database system to process said transactions.

27. A throttling device for interfacing disparate capacity computer database systems, comprising:

an input component for receiving transactions from a first database system at a first rate, wherein said transactions include data requests and/or data entries;

an output component for providing transactions to a second database system at a second rate; and

a buffer component coupled to said input component and said output component for delaying said transactions for a predetermined period of time when said first rate exceeds said second rate, wherein said predetermined period of time is a function of said second rate.

* * * * *